# VsxProtocolDriver C-Wrapper

3.3.2+ge9b14e0

*Driver package (C) to communicate with P+F SmartRunner devices via VSX protocol*

# 1 Introduction

The driver VsxProtocolDriver (VsxSdk) provides full access to the input and output data of the sensor. The driver connects to the sensor and handles communication in accordance with the communication protocol. The user can access functions for setting parameters on the sensor, retrieving parameter values from the sensor, and saving and loading entire parameter sets both locally and on the sensor. The user can also receive sensor data like images, 3D-data or lines. Each function also contains an error object from which information can be obtained in the event of an error in the function.

## 1.1 Supported devices

The official supported devices are the following:

- SmartRunner 3D (Stereo + ToF)
- SmartRunner 2D

### 1.2 Requirements

The driver is available for multiple architecture

- Windows 64 bit / 32 bit
- Linux AMD64, ARM64, ARM32

The main driver is based on the C# (.NET). There are wrapper for C and Python programming language available.

For usage the Microsoft .NET Runtime 6.0.x framework or higher must be installed (See `https://dotnet.microsoft.com/en-us/download/dotnet`).

**Important note:** There is also still support for .Net 5.0, but this will propably be dropped in the next version, as this release has reached end of life support by Microsoft.

## 2 Usage with C interface

The driver VsxProtocolDriver (VsxSdk) facilitates integration in a C- based programming environment.

The main driver is implemented in C# and requires .NET 6.0 or higher.

The functions of the C-wrapper can only be used synchronously.

### 2.1 Requirements

The driver is available as C library and header for multiple architecture

- Windows 64 bit / 32 bit
- Linux AMD64, ARM64, ARM32

The driver is based on the VsxProtocolDriver, which is based on C#. So for usage the Microsoft .NET Runtime 6.0.x framework or higher must be installed (See `https://dotnet.microsoft.com/en-us/download/dotnet`). There is also still support for .Net 5.0, but this will propably be dropped in the next version.

### 2.2 Installation

In order to use the SDK, the files are located inside the zipped driver. Unzip the files and select the correct architecture, e.g. "win-x64".

The driver package contains the following files:

- Header
  - PF.VsxProtocolDriver.WrapperNE.h (main header)
  - dnne.h (internal used header)
- Library
  - PF.VsxProtocolDriver.WrapperNE.lib (import library)
  - PF.VsxProtocolDriver.WrapperNE.dll/.so (dynamic library file)
- Referenced .net dlls
  - Multiple dlls located in `PF.VsxProtocolDriver.WrapperNE` folder

The remaining files are needed for the .net driver. These files and the dynamic library file must be located in the same folder as the executable to run.

## 2.3 Documentation

The `PF.VsxProtocolDriver.Wrapper` is a C# interface to communicate with VSX based sensors. It is based on the C# based `VsxProtocolDriverSync` driver. From the `PF.VsxProtocolDriver.Wrapper` an C-interface is build automatically.

### 2.3.1 Memory management

When the user requests string, images or other objects, the memory will be allocated on the C# side and needed to be release later with the "ReleaseXXX" function

```
#include "PF.VsxProtocolDriver.WrapperNE.h"

const char* version = NULL;
VsxStatusCode ret;
ret = vsx_GetLibraryVersion(&version);
if (ret == VSX_STATUS_SUCCESS)
{
    printf("Version %s\n", version);
    ret = vsx_ReleaseString(&version);
}
```

The reference of the object pointer (e.g. `const char* version`) must be set to `NULL`, before allocating the object. Otherwise the function will return an error. After allocation, the pointer is valid and can be used.

There are the following release functions available:

| Function | Object |
|---|---|
| vsx_ReleaseString | const char∗ |
| vsx_ReleaseSensor | VsxSystemHandle∗ |
| vsx_ReleaseDataContainer | VsxDataContainerHandle∗ |
| vsx_ReleaseImage | VsxImage∗ |
| vsx_ReleaseLine | VsxLine∗ |
| vsx_ReleaseDisparityDescriptor2 | VsxDisparityDescriptor2∗ |
| vsx_ReleaseTransformation | VsxTransformation∗ |
| vsx_ReleaseCaptureInformation | VsxCaptureInformation∗ |
| vsx_ReleaseOlr2CaptureInformation | VsxOlr2CaptureInformation∗ |
| vsx_ReleaseOlr2ModbusData | VsxOlr2ModbusData∗ |
| vsx_ReleaseTagList | VsxTagList∗ |
| vsx_ReleaseDevice | VsxDevice∗ |
| vsx_ReleaseDeviceList | VsxDeviceList∗ |
| vsx_ReleaseParameter | VsxParameter∗ |
| vsx_ReleaseParameterList | VsxParameterList∗ |
| vsx_ReleaseStatusItemList | VsxStatusItemList∗ |

When the release function is called, the reference of the object will be set back to `NULL`.

### 2.3.2 Error handling

Normally all functions return a `VsxStatusCode`. The enum is available in the `PF.VsxProtocolDriver.WrapperNE.h` header. A successful function return `VSX_STATUS_SUCCESS` (0). Errors starts with `VSX_STATUS_ERROR_XXX` and have a negative number.

To access the enum status code as text, the `vsx_GetErrorText` can be used:

```
#include "PF.VsxProtocolDriver.WrapperNE.h"

const char* error_text = NULL;
VsxStatusCode status_code = <call_another_function>(...);
VsxStatusCode ret = vsx_GetErrorText(status_code, &error_text);
if (ret == VSX_STATUS_SUCCESS)
{
    printf("Error %s\n", error_text);
    ret = vsx_ReleaseString(&error_text);
}
```

This will also return dynamic error as additional text, when available.

### 2.3.3 Support of dynamic container (including 3D data)

With the new available `Dynamic container` inside the VSX protocol, multiple images and data message can be send together. So now a container for e.g. SR3D stereo can contain a left image, right image and a disparity map. Other products can contain an image and a result message. For compatibility reason non dynamic container packages will be packed into a new dynamic container, when reaching the driver. So they can be used in the same manner.

The single message data (e.g. image, result) can be accessed by an `tag` name. This could be e.g. `LeftRaw`, `Image`. The naming is product specific.

The dynamic container are stored internally by a queue, which can be configured by the follwing function:

```
VsxStatusCode vsx_ResetDynamicContainerGrabber(VsxSystemHandle* vsx, int32_t bufferSize, int32_t
      startCondition, VsxStrategy strategy);
```

The buffer size allocates queue with a certain size. The strategy option change the behaviuour of new arriving dynamic containers. With `VSX_STRATEGY_DROP_OLDEST` the oldest are dropped, so that the queue is filled up with new dynamic container. With `VSX_STRATEGY_DROP_WRITE` the newest are dropped and the queue will only be filled until it is full.

The optional parameter `startCondition` is not used now.

To generate a queue, which always show the last transmitted `dynamic container` use `bufferSize=1` and `strategy=VSX_STRATEGY_DROP_OLDEST`.

To capture the next 5 trigger after the reset set `bufferSize=5` and `strategy=VSX_STRATEGY_DROP_WRITE`.

```
// allocatee sensor and connect it -> vsx

VsxStatusCode ret;
ret = vsx_ResetDynamicContainerGrabber(vsx, 1, -1, VSX_STRATEGY_DROP_OLDEST);

// Trigger sensor (SW or HW)

VsxDataContainerHandle* dch = NULL;
ret = vsx_GetDataContainer(vsx, &dch, 1000);
if (ret == VSX_STATUS_SUCCESS)
{
    ret = vsx_SaveData(dch, "Image", "Image.bmp");
    ret = vsx_ReleaseDataContainer(&dch);
}
```

### 2.3.4 Firmware Update

The firmware update will now be able to support both hardware platforms:

- Linux based system with rescue system
    - The firmware will be uploaded to the system.
    - Afterwards the sensor will start automatically to the rescue system and update the system
    - When the system starts again, the function will return.
- Texas Instruments (flash based system)
    - The update will be written into the flash memory.
    - The function will return afterswards.
    - The sensor must be started *manually* new by a power down-up cycle to get started with new firmware.

```
VsxStatusCode vsx_SendData(VsxSystemHandle* vsx, const char* fileName);
```

## 2.4 Functions

### 2.4.1 Common library functions

As described in the memory managment section, every string returned by a function must be freed by release string.

```
VsxStatusCode vsx_ReleaseString(const char** pString);
```

The version of driver can be accessed by the following function:

```
VsxStatusCode vsx_GetLibraryVersion(const char** version);
```

To get a text for a `VsxStatusCode` (including dynamic error text) use this function:

```
VsxStatusCode vsx_GetErrorText(int32_t error_code, const char** error_text);
```

### 2.4.2 Sensor handling

With the driver multiple sensor can be accessed.

To generate a tcp or serial sensor call the init function:

```
VsxStatusCode vsx_InitTcpSensor(VsxSystemHandle** pVsx, const char* ipAddress, const char* pluginName);
VsxStatusCode vsx_InitSerialSensor(VsxSystemHandle** pVsx, const char* serialPort, int32_t baudrate, const
      char* sensorType, VsxSerialConnectionType connectionType, const char* pluginName);
```

The connection can be tried to build up by the connect function. In the case of tcp sensor, where multiple devices accessed on the same ip address the call of `vsx_ConnectEx` with a `timeout_ms > 300000` might be useful.

```
VsxStatusCode vsx_Connect(VsxSystemHandle* vsx);
VsxStatusCode vsx_ConnectEx(VsxSystemHandle* vsx, int32_t timeout_ms);
```

> The arp cache of Windows delays the recognition of a new mac address on the same ip address. The process could take up to 30s.

To disconnect a sensor call:

```
VsxStatusCode vsx_Disconnect(VsxSystemHandle* vsx);
```

To release a sensor after usage call:

```
VsxStatusCode vsx_ReleaseSensor(VsxSystemHandle** vsx);
```

### 2.4.3 Sensor handling (Reconnect functions)

For testing it could be interesting to change a current connection e.g. to another ip address.

```
VsxStatusCode vsx_ReConnectTcpSensor(VsxSystemHandle* vsx, const char* ipAddress);
VsxStatusCode vsx_ReConnectSerialSensor(VsxSystemHandle* vsx, const char* serialPort, int32_t baudrate,
      VsxSerialConnectionType connectionType);
```

### 2.4.4 Sensor functions (excluding dynamic container)

The default timeout for standard function calls can be read and written

```
VsxStatusCode vsx_GetWaitTimeout(VsxSystemHandle* vsx, int32_t* result);
VsxStatusCode vsx_SetWaitTimeout(VsxSystemHandle* vsx, int32_t timeout_ms);
```

It is used by the functions declared in this paragraph.

The `vsx_TestSystem` is used by production. It gets a `command` and `inputValue` string as input and returns an `ouputValue`string and `status` code as result. It uses the default timeout.

```
VsxStatusCode vsx_TestSystem(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char**
      outputValue, int32_t* status);
```

The status code is `1`, if command call is valid, otherwise `0`.

The `vsx_TestSystemEx` is the same as above, but an indivudal timeout in ms could be set.

```
VsxStatusCode vsx_TestSystemEx(VsxSystemHandle* vsx, const char* command, const char* inputValue, const
      char** outputValue, int32_t* status, int32_t timeout_ms);
```

The status code is `1`, if command call is valid, otherwise `0`.

To change the ip address of the connected sensor, use the following function:

```
VsxStatusCode vsx_SetNetworkSettings(VsxSystemHandle* vsx, const char* ipAddress, const char* networkMask,
      const char* gateway);
```

### Parameter handling

To save the actual parameter set onto the sensor (and override the last saved parameter set) call
```
VsxStatusCode vsx_SaveParameterSetOnDevice(VsxSystemHandle* vsx);
```

To load the parameter set from the sensor (and override the actual parameter set) call
```
VsxStatusCode vsx_LoadParameterSetOnDevice(VsxSystemHandle* vsx);
```

To load the default parameter set from the sensor (and override the actual parameter set) call
```
VsxStatusCode vsx_LoadDefaultParameterSetOnDevice(VsxSystemHandle* vsx);
```

To upload or download the actual parameter set on the sensor the following functions are avaiable.
```
VsxStatusCode vsx_UploadParameterSet(VsxSystemHandle* vsx, const char* fileName);
VsxStatusCode vsx_DownloadParameterSet(VsxSystemHandle* vsx, const char* fileName);
```

To set and get a single parameter inside the parameter set, the following functions can be used.
```
VsxStatusCode vsx_SetSingleParameterValue(VsxSystemHandle* vsx, uint32_t settingsVersion, const char*
      configurationId, uint32_t configurationVersion, const char* parameterId, const char* value);
VsxStatusCode vsx_GetSingleParameterValue(VsxSystemHandle* vsx, uint32_t settingsVersion, const char*
      configurationId, uint32_t configurationVersion, const char* parameterId, const char** value);
```

The parameter structue has two layer. A configuration group (e.g. "Base") contains multiple parameters. The sensor itself contains one or multiple configuration groups. To make a serialization / deserialization of data possible, each layer has its own version number. With this in place, there is the possiblity to add new parameter / configuration, remove unused parameter and even change the unit of a parameter.

To define a parameter, the following parameters are needed:

settingsVersion -> Version number, which tells the senosr, which configurations are available configurationId -> Name of current configuration group configurationVersion -> Version number, which tells the senosr, which parameters are available in given configuration group parameterId -> Name of parameter id

An example would be the following parameter for the Smartrunner 3D Stereo sensor:

settingsVersion -> 2 configurationId -> "Base" configurationVersion -> 2 parameterId -> "ExposureTime"

When a parameter is set, with e.g. vsx_SetSingleParameterValue(vsx, 2, "Base", 2, "ExposureTime", "1000"); the serialzation will guarantee also in case of an sensor firmware upgrade, will still work.

### Send other data

To send data (like firmware, images or xml commands) to the sensor the following functions can be used:
```
VsxStatusCode vsx_UploadData(VsxSystemHandle* vsx, const char* fileName);
VsxStatusCode vsx_SendXmlMessageData(VsxSystemHandle* vsx, const char* xmlCommand);
VsxStatusCode vsx_SendFirmware(VsxSystemHandle* vsx, const char* fileName);
```

> Which kind of data is supported is sensor type specific

### 2.4.5 Log message handling

The log messages could also be buffered by the driver. With `vsx_ResetLogMessageGrabber` the handling of incoming data can be configured (see chapter "Support of dynamic container"). The type masked should be defined by the following bitmask:
```
LOGT_DBG = 0x01,
LOGT_INFO = 0x02,
LOGT_RESOK = 0x04,
LOGT_RESNOK = 0x08,
LOGT_WARN = 0x10,
LOGT_ERR = 0x20,
LOGT_CRIT = 0x40,
LOGT_ASSERT = 0x80,
LOGT_ALL = 0xFFFFFFFF
```

With `VSX_STRATEGY_DROP_OLDEST` the oldest are dropped, so that the queue is filled up with new dynamic container. With `VSX_STRATEGY_DROP_WRITE` the newest are dropped and the queue will only be filled until it is full.
```
VsxStatusCode vsx_ResetLogMessageGrabber(VsxSystemHandle* vsx, int32_t bufferSize, int32_t typeMask,
      VsxStrategy strategy);
VsxStatusCode vsx_GetLogMessage(VsxSystemHandle* vsx, const char** log, int32_t timeout_ms);
```

The `vsx_GetLogMessage` function tries to receive data in a certain time period (parameter `timeout_ms`) from the receiver queue.

### 2.4.6 Dynamic container handling

The dynamic container grabber is used to receive images, result and other data from the sensor.

#### configure, receive data and release

With `vsx_ResetDynamicContainerGrabber` the handling of incoming data can be configured (see chapter "Support of dynamic container").

```
VsxStatusCode vsx_ResetDynamicContainerGrabber(VsxSystemHandle* vsx, int32_t bufferSize, int32_t
      startCondition, VsxStrategy strategy);
```

The `vsx_GetDataContainer` function tries to receive data in a certain time period (parameter `timeout_ms`) from the receiver queue.

```
VsxStatusCode vsx_GetDataContainer(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t timeout_ms);
```

Now the data of the container be accessed. Afterwards the container must be freed by the following function:

```
VsxStatusCode vsx_ReleaseDataContainer(VsxDataContainerHandle** dch);
```

#### Tag list

When a dynamic container is received, the data of the container could be listed. Therefor `vsx_GetTagList` could receive an array of tag names. This must be deleted afterwards with `vsx_ReleaseTagList`.

```
VsxStatusCode vsx_GetTagList(VsxDataContainerHandle* dch, VsxTagList** tagList)
VsxStatusCode vsx_ReleaseTagList(VsxTagList** pTagList)
```

> Somtimes tags are generated on the fly, e.g. for 3D data from disparity values. These are not listed.

#### Access data

When a dynamic container is received, the data can be accessed. The data inside the dynamic container can be accessed by the `tag` name, which are defined by the sensor type. Examples are `Image`, `LeftRaw`, `Result`.

With `vsx_SaveData` you can save data into a file on the PC. The `tag` and the `filename` must be specified. The file ending will be used to define e.g. the image encoding (`.bmp`, `.png`). When using `tag="*"` an `.zip` as file ending the complete dynamic container will be saved compressed file.

```
VsxStatusCode vsx_SaveData(VsxDataContainerHandle* dch, const char* tag, const char* fileName);
```

For images there is also the possibility to generate a memory access. This can be done with `vsx_GetImage`and delivers a `VsxImage`, where the raw data pointer and attributes like height and width are defined. After usage the memory must be released by `vsx_ReleaseImage`.

```
VsxStatusCode vsx_GetImage(VsxDataContainerHandle* dch, const char* imageTag, VsxImage** imageData);
VsxStatusCode vsx_ReleaseImage(VsxImage** pImage);
```

To save point cloud data, all three data layer must be specified. The supported data format is only the `.pcd` (Point Cloud Data) file format.

```
VsxStatusCode vsx_Save3DPointCloudData(VsxDataContainerHandle* dch, const char* point_x_Id, const char*
      point_y_Id, const char* point_z_Id, const char* fileName);
```

For result data the whole xml string (`vsx_GetResultXml`) can be received. To receive a single value from the xml an xml path expression must be given. The return value will be converted to the selected data type (string, int, long or double)

```
VsxStatusCode vsx_GetResultXml(VsxDataContainerHandle* dch, const char* resultId, const char** result);
VsxStatusCode vsx_GetResultElementString(VsxDataContainerHandle* dch, const char* resultId, const char*
      xPath, const char** result);
VsxStatusCode vsx_GetResultElementInt32(VsxDataContainerHandle* dch, const char* resultId, const char*
      xPath, int32_t* result);
VsxStatusCode vsx_GetResultElementInt64(VsxDataContainerHandle* dch, const char* resultId, const char*
      xPath, int64_t* result);
VsxStatusCode vsx_GetResultElementDouble(VsxDataContainerHandle* dch, const char* resultId, const char*
      xPath, double* result);
```

### 2.4.7 Display device information

To read out the current device information call the following function. There is also a release function, to free up the memory by the driver.

```
VsxStatusCode vsx_GetCurrentDeviceInformation(VsxSystemHandle* vsx, VsxDevice** deviceData);
VsxStatusCode vsx_ReleaseDevice(VsxDevice** pDevice);
```

Send an UDP request to find sensor in the network. It returns a list, with the information. There is also a release function, to free up the memory by the driver.

```
VsxStatusCode vsx_GetUdpDeviceList(VsxDeviceList** deviceListData);
VsxStatusCode vsx_ReleaseDeviceList(VsxDeviceList** pDeviceList);
```

# 3 Examples

In the following the usage of the `VsxProtocolDriver` is shown with a short code example.

The complete examples can be found as a CMake project in the `C\example\` subfolder. It support the detection of different sensors and show the parametrization and the grabbing of data from the sensor.

```
#include "PF.VsxProtocolDriver.WrapperNE.h"
#include <stdio.h>

void print_error(VsxStatusCode code)
{
    const char* error_text = NULL;
    VsxStatusCode ret;
    ret = vsx_GetErrorText(code, &error_text);
    if (ret == VSX_STATUS_SUCCESS)
    {
        printf("Error code %s\n", error_text);
        ret = vsx_ReleaseString(&error_text);
    }
}

int main(int argc, char** argv) {
    VsxDeviceList* devList = NULL;
    VsxStatusCode ret;
    VsxSystemHandle* vsx = NULL;

    ret = vsx_GetUdpDeviceList(&devList);

    if (ret != VSX_STATUS_SUCCESS)
    {
        return -1;
    }
    if (devList->length > 0)
    {
        // create a new VsxProtocolDriver instance
        VsxDevice dev = devList->devices[0];
        printf("Device found: %s %s\n", dev.sensorType, dev.ipAddress);
        ret = vsx_InitTcpSensor(&vsx, dev.ipAddress, "");
    }
    else //use fix ip address
    {
        // create a new VsxProtocolDriver instance with fix ip address
        ret = vsx_InitTcpSensor(&vsx, "192.168.2.4", "");
    }
    if (ret != VSX_STATUS_SUCCESS)
    {
        print_error(ret);
        return -2;
    }

    // Connect with device
    ret = vsx_Connect(vsx);
    if (ret != VSX_STATUS_SUCCESS)
    {
        print_error(ret);
        ret = vsx_ReleaseSensor(&vsx);
        return -3;
    }

    // Get the current device information
    VsxDevice* device = NULL;
    ret = vsx_GetDeviceInformation(vsx, &device);
    if (ret == VSX_STATUS_SUCCESS)
    {
        printf("Actual IP from device %s\n", device->ipAddress);
        ret = vsx_ReleaseDevice(&device);
        if (ret != VSX_STATUS_SUCCESS)
        {
            print_error(ret);
            ret = vsx_ReleaseSensor(&vsx);
            return -5;
        }
    }
    else
    {
        print_error(ret);
        ret = vsx_ReleaseSensor(&vsx);
        return -4;
    }

    // Release sensor instance
    ret = vsx_ReleaseSensor(&vsx);
    if (ret != VSX_STATUS_SUCCESS)
    {
        print_error(ret);
        return -5;
    }

    return 0;
}
```

# 4 Device parameter

In this chapter some information about the structure of the device parameters shall be given.

The device parameters are organized in two levels. The first level includes one or more configuration groups. Each of these configuration groups in turn contains one or more parameters. To uniquely identify parameters, each configuration has a unique Id. Each parameter also contains an Id that is unique within its configuration.

In order to keep different firmware versions compatible with each other, an additional versioning exists. This comprises on the one hand a settings version, which determines, which configurations are present up-to-date, and a configuration version, which determines which parameters are present at the moment in this configuration. If changes are made to configurations or parameters, the respective version number is increased.

Four arguments are hence required to uniquely define a device parameter:

- *settingsVersion*: Version number, which tells the device which configurations are available
- *configurationId*: Id of the current configuration group
- *configurationVersion*: Version number, which tells the device which parameters are available whithin the current configuration group and how they are handled
- *parameterId*: Id of the current parameter

In order to know the individual parameters with their ids and versions, files for all supported sensor types and their various firmware versions are stored in a source file in the example subfolder named with `<sensor_name>ParameterIdentifier`. The required informations can be taken from these files.

**Example**: The value of the following parameter for the Smartrunner 3-D device:

- *settingsVersion*: 2
- *configurationId*: "Base"
- *configurationVersion*: 2
- *parameterId*: "ExposureTime"

can be received using the driver via the function `GetSingleParameterValue(settingsVersion:2, configId:"Base", configVersion:2, parameterId:"ExposureTime").`

**Additional notes:**

- if a configuration or parameter does not contain a version attribute, use the default value of "1".
- in addition to the information on version and Id, the xml files also contain further information on the parameters such as name, value range, etc.
- to trigger event parameters these must be set to a value of "1".
- for the Smartrunner 2-D, only a part of the parameters is listed in the corresponding xml file. Only these parameters should be used for parameterization of the device.

# 5 Changelog

This is the changelog for the C implementation of the VsxProtocolDriver. It is based on the .NET implementation (C#) of the VsxProtocolDriver. Please use also the .NET documenatation for additional informations about the release.

V3.3.2

- based on VsxProtocolDriver 3.3.2

V3.3.1

- based on VsxProtocolDriver 3.3.1

V3.3.0

- based on VsxProtocolDriver 3.3.0

V3.2.1

- based on VsxProtocolDriver 3.2.1
- Add access to write and read single parameter with "double" and "int32_t" type (instead of only string as before)

V3.1.5

- based on VsxProtocolDriver 3.1.5
  - Modified "Olr2CaptureInformation" data structure (incompatible with V3.1.0 and following, only for Olr2!)

V3.1.4

- Fix memory leak inside line data allocation

V3.1.3

- based on VsxProtocolDriver 3.1.3
  - use given ip address instead udp response to connect

V3.1.2

- based on VsxProtocolDriver 3.1.2
- Receive also "ApplicationResultData" as result
- Correct handling of VsxLine (when Quality is missing)
- Usage of UTF-8 for string encoding

V3.1.1

- based on VsxProtocolDriver 3.1.1
- Add ".VsxLine" (xml-based) as new and default line format

V3.1.0

- based on VsxProtocolDriver 3.1.0
- Add function "vsx_SendSessionKeepAlive" (reply to timeout announcement message)
- Added support for "Olr2ModbusData" & "Olr2CaptureInformation"

V3.0.6

- based on VsxProtocolDriver 3.0.6
  - Correct saving of images in "bmp" & "png" format again (introduced in 3.0.5)

V3.0.5

- based on VsxProtocolDriver 3.0.5
  - Allow saving of images in "bmp" format again
- Change "Parameter" struct handling, values are set now by dedicated function, e.g. "vsx_SetSingle↩ ParameterString"
- Add missing "VsxImageData2Format" enum values in header:
  - VSX_IMAGE_DATA2_FORMAT_COORD3D_A32F, VSX_IMAGE_DATA2_FORMAT_COORD3D_B32F, VSX_IMAGE_DATA2_FORMAT_COORD3D_C32F
- Add comments to explain, which value type to use for "VsxParameterValueType" & "VsxStatusItemValueType"

V3.0.4

- Remove "InitTcpSensorEx" & "ReConnectTcpDeviceEx" function from 3.0.2/3.0.3 again (not needed, if direct UDP from sensor supported)

- based on VsxProtocolDriver 3.0.4

V3.0.3

- Add "ReConnectTcpDeviceEx" function to support setting of port number
- Correct V3.0.2, where "ReConnectTcpDevice" used already new port attribute.

V3.0.2

- Add "InitTcpSensorEx" function to support setting of port number

V3.0.1

- Correct handling for 64 bit values for (really) old compiler

V3.0.0

- based on VsxProtocolDriver 3.0.0
- First support for encrypted login
- Changed function call of "vsx_ResetDynamicContainerGrabber" (removed one unused parameter)

# 6 Class Index

## 6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

**_VsxCaptureInformation**
**Contains information about image capture** 13

**_VsxDataContainerHandle**
**Structure to use for a data container instance** 15

**_VsxDevice**
**Declare device informations** 15

**_VsxDeviceList**
**List of devices** 17

**_VsxDisparityDescriptor2**
**Disparity descriptor to calculate 3D data from disparity map** 18

**_VsxImage**
**Declaration of image data** 19

**_VsxLineCoordinate**
**Declare coordinate point of line** 21

# 7 File Index

## 7.1 File List

Here is a list of all files with brief descriptions:

# 8 Class Documentation

## 8.1 _VsxCaptureInformation Struct Reference

Contains information about image capture.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

**Public Attributes**

- LOCAL_UINT64_T triggerCounter
- LOCAL_UINT64_T parameterId
- LOCAL_UINT64_T jobId
- LOCAL_INT64_T rotaryEncoder
- LOCAL_UINT64_T frameCounter
- LOCAL_UINT64_T timestamp
- unsigned int exposureTime
- unsigned int gain
- unsigned char illumination
- unsigned char triggerSource

### 8.1.1 Detailed Description

Contains information about image capture.

### 8.1.2 Member Data Documentation

#### triggerCounter

`LOCAL_UINT64_T _VsxCaptureInformation::triggerCounter`

#### parameterId

`LOCAL_UINT64_T _VsxCaptureInformation::parameterId`

#### jobId

`LOCAL_UINT64_T _VsxCaptureInformation::jobId`

#### rotaryEncoder

`LOCAL_INT64_T _VsxCaptureInformation::rotaryEncoder`

#### frameCounter

`LOCAL_UINT64_T _VsxCaptureInformation::frameCounter`

#### timestamp

`LOCAL_UINT64_T _VsxCaptureInformation::timestamp`

**exposureTime**

`unsigned int _VsxCaptureInformation::exposureTime`

**gain**

`unsigned int _VsxCaptureInformation::gain`

**illumination**

`unsigned char _VsxCaptureInformation::illumination`

**triggerSource**

`unsigned char _VsxCaptureInformation::triggerSource`

## 8.2 _VsxDataContainerHandle Struct Reference

Structure to use for a data container instance.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- int handle

  *Handle to data container instance.*

### 8.2.1 Detailed Description

Structure to use for a data container instance.

### 8.2.2 Member Data Documentation

**handle**

`int _VsxDataContainerHandle::handle`

Handle to data container instance.

## 8.3 _VsxDevice Struct Reference

Declare device informations.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- const char ∗ ipAddress
- const char ∗ networkMask
- const char ∗ gateway
- const char ∗ macAddress
- const char ∗ firmwareVersion
- const char ∗ sensorType
- const char ∗ sensorName
- int busy
- int deviceVsxVersionMajor
- int deviceVsxVersionMinor
- const char ∗ comPort
- int baudrate
- const char ∗ headAddress
- int isLoginNeeded

## 8.3.1 Detailed Description

Declare device informations.

## 8.3.2 Member Data Documentation

### ipAddress

`const char* _VsxDevice::ipAddress`

### networkMask

`const char* _VsxDevice::networkMask`

### gateway

`const char* _VsxDevice::gateway`

### macAddress

`const char* _VsxDevice::macAddress`

### firmwareVersion

`const char* _VsxDevice::firmwareVersion`

### sensorType

`const char* _VsxDevice::sensorType`

### sensorName

`const char* _VsxDevice::sensorName`

### busy

`int _VsxDevice::busy`

### deviceVsxVersionMajor

`int _VsxDevice::deviceVsxVersionMajor`

### deviceVsxVersionMinor

`int _VsxDevice::deviceVsxVersionMinor`

### comPort

`const char* _VsxDevice::comPort`

### baudrate

`int _VsxDevice::baudrate`

### headAddress

`const char* _VsxDevice::headAddress`

### isLoginNeeded

`int _VsxDevice::isLoginNeeded`

## 8.4 _VsxDeviceList Struct Reference

List of devices.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- int length
- const VsxDevice * devices

### 8.4.1 Detailed Description

List of devices.

### 8.4.2 Member Data Documentation

#### length

`int _VsxDeviceList::length`

#### devices

`const VsxDevice* _VsxDeviceList::devices`

## 8.5 _VsxDisparityDescriptor2 Struct Reference

Disparity descriptor to calculate 3D data from disparity map.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

#### Public Attributes

- double focalLength
- double principalPointU
- double principalPointV
- double baseline
- double offsetLeftRectifiedToDisparityU
- double offsetLeftRectifiedToDisparityV

### 8.5.1 Detailed Description

Disparity descriptor to calculate 3D data from disparity map.

### 8.5.2 Member Data Documentation

#### focalLength

`double _VsxDisparityDescriptor2::focalLength`

#### principalPointU

`double _VsxDisparityDescriptor2::principalPointU`

#### principalPointV

```
double _VsxDisparityDescriptor2::principalPointV
```

#### baseline

```
double _VsxDisparityDescriptor2::baseline
```

#### offsetLeftRectifiedToDisparityU

```
double _VsxDisparityDescriptor2::offsetLeftRectifiedToDisparityU
```

#### offsetLeftRectifiedToDisparityV

```
double _VsxDisparityDescriptor2::offsetLeftRectifiedToDisparityV
```

## 8.6 _VsxImage Struct Reference

Declaration of image data.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- void ∗ rawdata
- VsxImageData2Format format
- int width
- int height
- int linePitch
- LOCAL_INT64_T frameCounter
- double coordinateScale
- double coordinateOffset
- double axisMin
- double axisMax
- double invalidDataValue

### 8.6.1 Detailed Description

Declaration of image data.

### 8.6.2 Member Data Documentation

#### rawdata

```
void* _VsxImage::rawdata
```

### format

`VsxImageData2Format _VsxImage::format`

### width

`int _VsxImage::width`

### height

`int _VsxImage::height`

### linePitch

`int _VsxImage::linePitch`

### frameCounter

`LOCAL_INT64_T _VsxImage::frameCounter`

### coordinateScale

`double _VsxImage::coordinateScale`

### coordinateOffset

`double _VsxImage::coordinateOffset`

### axisMin

`double _VsxImage::axisMin`

### axisMax

`double _VsxImage::axisMax`

### invalidDataValue

`double _VsxImage::invalidDataValue`

## 8.7 _VsxLineCoordinate Struct Reference

Declare coordinate point of line.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- float c

  *Image column [px].*
- float x

  *Position X direction [mm].*
- float y

  *Position Y direction [mm].*
- float z

  *Position Z direction [mm].*
- float q

  *Quality value [0..100%].*
- float i

  *Intensity [grayvalues].*

### 8.7.1 Detailed Description

Declare coordinate point of line.

### 8.7.2 Member Data Documentation

**c**

`float _VsxLineCoordinate::c`

Image column [px].

**x**

`float _VsxLineCoordinate::x`

Position X direction [mm].

**y**

`float _VsxLineCoordinate::y`

Position Y direction [mm].

**z**

`float _VsxLineCoordinate::z`

Position Z direction [mm].

**q**

`float _VsxLineCoordinate::q`

Quality value [0..100%].

**i**

`float _VsxLineCoordinate::i`

Intensity [grayvalues].

## 8.8 _VsxLineData Struct Reference

Declare a line package.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- VsxLineCoordinate ** lines
- unsigned short format
- unsigned short width
- unsigned short countLines
- unsigned short frameCounter
- float minX
- float maxX
- float minZ
- float maxZ

### 8.8.1 Detailed Description

Declare a line package.

### 8.8.2 Member Data Documentation

**lines**

`VsxLineCoordinate** _VsxLineData::lines`

### format

`unsigned short _VsxLineData::format`

### width

`unsigned short _VsxLineData::width`

### countLines

`unsigned short _VsxLineData::countLines`

### frameCounter

`unsigned short _VsxLineData::frameCounter`

### minX

`float _VsxLineData::minX`

### maxX

`float _VsxLineData::maxX`

### minZ

`float _VsxLineData::minZ`

### maxZ

`float _VsxLineData::maxZ`

## 8.9 _VsxOlr2CaptureInformation Struct Reference

Contains information about image capture.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

**Public Attributes**

- LOCAL_UINT64_T frameCounter
- LOCAL_UINT64_T triggerCounter
- double currentPosition
- LOCAL_UINT64_T ioState
- LOCAL_UINT64_T timestamp
- unsigned int ImaExposureTime1
- unsigned int ImaExposureTime2
- unsigned int ImbExposureTime1
- unsigned int ImbExposureTime2
- unsigned short ImaRoiOffsetX
- unsigned short ImaRoiLengthX
- unsigned short ImaRoiOffsetZ
- unsigned short ImaRoiLengthZ
- unsigned short ImbRoiOffsetX
- unsigned short ImbRoiLengthX
- unsigned short ImbRoiOffsetZ
- unsigned short ImbRoiLengthZ
- unsigned short autoTriggerFrameRate
- unsigned char triggerSource

### 8.9.1 Detailed Description

Contains information about image capture.

### 8.9.2 Member Data Documentation

**frameCounter**

`LOCAL_UINT64_T _VsxOlr2CaptureInformation::frameCounter`

**triggerCounter**

`LOCAL_UINT64_T _VsxOlr2CaptureInformation::triggerCounter`

**currentPosition**

`double _VsxOlr2CaptureInformation::currentPosition`

**ioState**

`LOCAL_UINT64_T _VsxOlr2CaptureInformation::ioState`

**timestamp**

`LOCAL_UINT64_T _VsxOlr2CaptureInformation::timestamp`

### lmaExposureTime1

`unsigned int _VsxOlr2CaptureInformation::lmaExposureTime1`

### lmaExposureTime2

`unsigned int _VsxOlr2CaptureInformation::lmaExposureTime2`

### lmbExposureTime1

`unsigned int _VsxOlr2CaptureInformation::lmbExposureTime1`

### lmbExposureTime2

`unsigned int _VsxOlr2CaptureInformation::lmbExposureTime2`

### lmaRoiOffsetX

`unsigned short _VsxOlr2CaptureInformation::lmaRoiOffsetX`

### lmaRoiLengthX

`unsigned short _VsxOlr2CaptureInformation::lmaRoiLengthX`

### lmaRoiOffsetZ

`unsigned short _VsxOlr2CaptureInformation::lmaRoiOffsetZ`

### lmaRoiLengthZ

`unsigned short _VsxOlr2CaptureInformation::lmaRoiLengthZ`

### lmbRoiOffsetX

`unsigned short _VsxOlr2CaptureInformation::lmbRoiOffsetX`

### lmbRoiLengthX

`unsigned short _VsxOlr2CaptureInformation::lmbRoiLengthX`

#### lmbRoiOffsetZ

`unsigned short _VsxOlr2CaptureInformation::lmbRoiOffsetZ`

#### lmbRoiLengthZ

`unsigned short _VsxOlr2CaptureInformation::lmbRoiLengthZ`

#### autoTriggerFrameRate

`unsigned short _VsxOlr2CaptureInformation::autoTriggerFrameRate`

#### triggerSource

`unsigned char _VsxOlr2CaptureInformation::triggerSource`

## 8.10 _VsxOlr2ModbusData Struct Reference

Contains information about image capture.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

#### Public Attributes

- unsigned short activationTimer
- unsigned short compareBuffer
- unsigned short targetPosition
- unsigned short robotData [13]

### 8.10.1 Detailed Description

Contains information about image capture.

### 8.10.2 Member Data Documentation

#### activationTimer

`unsigned short _VsxOlr2ModbusData::activationTimer`

#### compareBuffer

`unsigned short _VsxOlr2ModbusData::compareBuffer`

#### targetPosition

`unsigned short _VsxOlr2ModbusData::targetPosition`

#### robotData

`unsigned short _VsxOlr2ModbusData::robotData[13]`

## 8.11 _VsxParameter Struct Reference

Declares parameter.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

#### Public Attributes

- const char ∗ valueString
- int valueInt
- double valueDouble
- VsxParameterValueType valueType
- const char ∗ name
- const char ∗ parameterId
- const char ∗ configId
- int configVersion
- int settingsVersion
- int enumItemListLength
- const VsxParameterEnumItem ∗ enumItemList

### 8.11.1 Detailed Description

Declares parameter.

### 8.11.2 Member Data Documentation

#### valueString

`const char* _VsxParameter::valueString`

#### valueInt

`int _VsxParameter::valueInt`

#### valueDouble

`double _VsxParameter::valueDouble`

### valueType

`VsxParameterValueType _VsxParameter::valueType`

### name

`const char* _VsxParameter::name`

### parameterId

`const char* _VsxParameter::parameterId`

### configId

`const char* _VsxParameter::configId`

### configVersion

`int _VsxParameter::configVersion`

### settingsVersion

`int _VsxParameter::settingsVersion`

### enumItemListLength

`int _VsxParameter::enumItemListLength`

### enumItemList

`const VsxParameterEnumItem* _VsxParameter::enumItemList`

## 8.12 _VsxParameterEnumItem Struct Reference

Single item of a parameter enum.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- const char * id
- const char * name

### 8.12.1 Detailed Description

Single item of a parameter enum.

### 8.12.2 Member Data Documentation

**id**

`const char* _VsxParameterEnumItem::id`

**name**

`const char* _VsxParameterEnumItem::name`

## 8.13 _VsxParameterList Struct Reference

List of parameter.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

**Public Attributes**

- int length
- const VsxParameter * parameters

### 8.13.1 Detailed Description

List of parameter.

### 8.13.2 Member Data Documentation

**length**

`int _VsxParameterList::length`

**parameters**

`const VsxParameter* _VsxParameterList::parameters`

## 8.14 _VsxStatusItem Struct Reference

Declaration of status item.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

## Public Attributes

- const char ∗ valueString
- int valueInt
- double valueDouble
- VsxStatusItemValueType valueType
- const char ∗ name
- const char ∗ statusItemId
- const char ∗ configurationClass
- int configVersion
- int settingsVersion
- LOCAL_UINT64_T time
- LOCAL_UINT64_T sensorTime

### 8.14.1 Detailed Description

Declaration of status item.

### 8.14.2 Member Data Documentation

#### valueString

`const char* _VsxStatusItem::valueString`

#### valueInt

`int _VsxStatusItem::valueInt`

#### valueDouble

`double _VsxStatusItem::valueDouble`

#### valueType

`VsxStatusItemValueType _VsxStatusItem::valueType`

#### name

`const char* _VsxStatusItem::name`

#### statusItemId

`const char* _VsxStatusItem::statusItemId`

#### configurationClass

`const char* _VsxStatusItem::configurationClass`

#### configVersion

`int _VsxStatusItem::configVersion`

#### settingsVersion

`int _VsxStatusItem::settingsVersion`

#### time

`LOCAL_UINT64_T _VsxStatusItem::time`

#### sensorTime

`LOCAL_UINT64_T _VsxStatusItem::sensorTime`

## 8.15 _VsxStatusItemList Struct Reference

List of status items.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- int length
- const VsxStatusItem * statusItems

### 8.15.1 Detailed Description

List of status items.

### 8.15.2 Member Data Documentation

#### length

`int _VsxStatusItemList::length`

#### statusItems

`const VsxStatusItem* _VsxStatusItemList::statusItems`

## 8.16 _VsxSystemHandle Struct Reference

Structure to use for sensor instance.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

#### Public Attributes

- int handle

  *Handle to sensor instance.*

### 8.16.1 Detailed Description

Structure to use for sensor instance.

### 8.16.2 Member Data Documentation

#### handle

`int _VsxSystemHandle::handle`

Handle to sensor instance.

## 8.17 _VsxTagList Struct Reference

List of all possible tags inside a dynamic container.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

#### Public Attributes

- int length
- const char ** tags

### 8.17.1 Detailed Description

List of all possible tags inside a dynamic container.

### 8.17.2 Member Data Documentation

#### length

`int _VsxTagList::length`

#### tags

`const char** _VsxTagList::tags`

## 8.18 _VsxTransformation Struct Reference

Transformation containg translation and quaternion.

`#include <PF.VsxProtocolDriver.WrapperNE.h>`

### Public Attributes

- double translationTX
- double translationTY
- double translationTZ
- double quaternionQ0
- double quaternionQ1
- double quaternionQ2
- double quaternionQ3

### 8.18.1 Detailed Description

Transformation containg translation and quaternion.

### 8.18.2 Member Data Documentation

#### translationTX

`double _VsxTransformation::translationTX`

#### translationTY

`double _VsxTransformation::translationTY`

#### translationTZ

`double _VsxTransformation::translationTZ`

#### quaternionQ0

```
double _VsxTransformation::quaternionQ0
```

#### quaternionQ1

```
double _VsxTransformation::quaternionQ1
```

#### quaternionQ2

```
double _VsxTransformation::quaternionQ2
```

#### quaternionQ3

```
double _VsxTransformation::quaternionQ3
```

# 9 File Documentation

## 9.1 01_Introduction.md File Reference

## 9.2 02_Usage_with_c.md File Reference

## 9.3 03_Examples.md File Reference

## 9.4 04_Device_parameter.md File Reference

## 9.5 05_Changelog.md File Reference

## 9.6 dnne.h File Reference

### Macros

- #define DNNE_LINUX
- #define DNNE_API __attribute__((__visibility__("default")))
- #define DNNE_CALLTYPE
- #define DNNE_CALLTYPE_CDECL
- #define DNNE_CALLTYPE_STDCALL
- #define DNNE_CALLTYPE_THISCALL
- #define DNNE_CALLTYPE_FASTCALL
- #define DNNE_STR(s) s
- #define DNNE_WCHAR uint16_t
- #define DNNE_SUCCESS 0
- #define DNNE_EXTERN_C

### Typedefs

- typedef void(DNNE_CALLTYPE ∗ failure_fn) (enum failure_type type, int error_code)

### Enumerations

- enum failure_type { failure_load_runtime = 1 , failure_load_export }

### Functions

- DNNE_API void DNNE_CALLTYPE set_failure_callback (failure_fn cb)
- DNNE_API void DNNE_CALLTYPE preload_runtime (void)
- DNNE_API int DNNE_CALLTYPE try_preload_runtime (void)
- DNNE_API void dnne_abort (enum failure_type type, int error_code)

## 9.6.1 Macro Definition Documentation

### DNNE_LINUX

```
#define DNNE_LINUX
```

### DNNE_API

```
#define DNNE_API __attribute__((__visibility__("default")))
```

### DNNE_CALLTYPE

```
#define DNNE_CALLTYPE
```

### DNNE_CALLTYPE_CDECL

```
#define DNNE_CALLTYPE_CDECL
```

### DNNE_CALLTYPE_STDCALL

```
#define DNNE_CALLTYPE_STDCALL
```

### DNNE_CALLTYPE_THISCALL

```
#define DNNE_CALLTYPE_THISCALL
```

### DNNE_CALLTYPE_FASTCALL

```
#define DNNE_CALLTYPE_FASTCALL
```

### DNNE_STR

```
#define DNNE_STR(
              s ) s
```

### DNNE_WCHAR

```
#define DNNE_WCHAR uint16_t
```

### DNNE_SUCCESS

```
#define DNNE_SUCCESS 0
```

### DNNE_EXTERN_C

```
#define DNNE_EXTERN_C
```

## 9.6.2 Typedef Documentation

### failure_fn

```
typedef void(DNNE_CALLTYPE * failure_fn) (enum failure_type type, int error_code)
```

## 9.6.3 Enumeration Type Documentation

### failure_type

```
enum failure_type
```

**Enumerator**

| | |
|---|---|
| failure_load_runtime | |
| failure_load_export | |

## 9.6.4 Function Documentation

### set_failure_callback()

```
DNNE_API void DNNE_CALLTYPE set_failure_callback (
            failure_fn cb )
```

### preload_runtime()

```
DNNE_API void DNNE_CALLTYPE preload_runtime (
            void  )
```

### try_preload_runtime()

```
DNNE_API int DNNE_CALLTYPE try_preload_runtime (
            void  )
```

### dnne_abort()

```
DNNE_API void dnne_abort (
            enum failure_type type,
            int error_code )  [extern]
```

## 9.7 dnne.h

Go to the documentation of this file.

```
// Copyright 2020 Aaron R Robinson
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is furnished
// to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
// OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
// SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#ifndef __SRC_PLATFORM_DNNE_H__
#define __SRC_PLATFORM_DNNE_H__

// Define our platform
#ifdef _WIN32
    #define DNNE_WINDOWS
#elif defined(__APPLE__)
    #define DNNE_OSX
#elif defined(__FreeBSD__)
    #define DNNE_FREEBSD
#else
    #define DNNE_LINUX
#endif

// Define some platform macros
#ifdef DNNE_WINDOWS
    #define DNNE_API __declspec(dllexport)
    #define DNNE_CALLTYPE __stdcall
    #define DNNE_CALLTYPE_CDECL __cdecl
    #define DNNE_CALLTYPE_STDCALL __stdcall
    #define DNNE_CALLTYPE_THISCALL __thiscall
    #define DNNE_CALLTYPE_FASTCALL __fastcall
    #define _DNNE_STR(s1) L ## s1
    #define DNNE_STR(s) _DNNE_STR(s)
    #define DNNE_WCHAR wchar_t
#else
```

```
    #define DNNE_API __attribute__((__visibility__("default")))
    #define DNNE_CALLTYPE
    #define DNNE_CALLTYPE_CDECL
    #ifdef __i386__
        #define DNNE_CALLTYPE_STDCALL __attribute__((stdcall))
        #define DNNE_CALLTYPE_THISCALL __attribute__((thiscall))
        #define DNNE_CALLTYPE_FASTCALL __attribute__((fastcall))
    #else
        #define DNNE_CALLTYPE_STDCALL
        #define DNNE_CALLTYPE_THISCALL
        #define DNNE_CALLTYPE_FASTCALL
    #endif
    #define DNNE_STR(s) s
    #define DNNE_WCHAR uint16_t
#endif

// Override the DNNE_API macro.
// This is typically used to dictate the export semantics of functions.
#ifdef DNNE_API_OVERRIDE
    #undef DNNE_API
    #define DNNE_API DNNE_API_OVERRIDE
#endif

//
// Public exports
//
#define DNNE_SUCCESS 0

enum failure_type
{
    failure_load_runtime = 1,
    failure_load_export,
};
typedef void (DNNE_CALLTYPE* failure_fn)(enum failure_type type, int error_code);

#ifdef __cplusplus
    #define DNNE_EXTERN_C extern "C"
    DNNE_EXTERN_C
    {
#else
    #define DNNE_EXTERN_C
#endif

// Provide a callback for any catastrophic failures.
// The provided callback will be the last call prior to a rude-abort of the process.
// See dnne_abort().
DNNE_API void DNNE_CALLTYPE set_failure_callback(failure_fn cb);

// Preload the runtime.
// The runtime is lazily loaded whenever the first export is called. This function
// preloads the runtime independent of calling any export and avoids the startup
// cost associated with calling an export for the first time.
// If the runtime fails to load, dnne_abort() will be called.
DNNE_API void DNNE_CALLTYPE preload_runtime(void);

// Attempt to preload the runtime.
// The runtime is lazily loaded whenever the first export is called. This function
// preloads the runtime independent of calling any export and avoids the startup
// cost associated with calling an export for the first time.
// If the runtime fails to load, an error code will be returned.
DNNE_API int DNNE_CALLTYPE try_preload_runtime(void);

// Users can override DNNE’s rude-abort behavior by providing their own dnne_abort() at link time.
// It is expected this function will not return. If it does return, the behavior is undefined.
extern DNNE_API void dnne_abort(enum failure_type type, int error_code);

#ifdef __cplusplus
    }
#endif

#endif // __SRC_PLATFORM_DNNE_H__
```

## 9.8 PF.VsxProtocolDriver.WrapperNE.h File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <dnne.h>
```

## Classes

- struct _VsxSystemHandle

  *Structure to use for sensor instance.*
- struct _VsxDataContainerHandle

  *Structure to use for a data container instance.*
- struct _VsxImage

  *Declaration of image data.*
- struct _VsxLineCoordinate

  *Declare coordinate point of line.*
- struct _VsxLineData

  *Declare a line package.*
- struct _VsxDisparityDescriptor2

  *Disparity descriptor to calculate 3D data from disparity map.*
- struct _VsxTransformation

  *Transformation containg translation and quaternion.*
- struct _VsxCaptureInformation

  *Contains information about image capture.*
- struct _VsxOlr2CaptureInformation

  *Contains information about image capture.*
- struct _VsxOlr2ModbusData

  *Contains information about image capture.*
- struct _VsxTagList

  *List of all possible tags inside a dynamic container.*
- struct _VsxDevice

  *Declare device informations.*
- struct _VsxDeviceList

  *List of devices.*
- struct _VsxParameterEnumItem

  *Single item of a parameter enum.*
- struct _VsxParameter

  *Declares parameter.*
- struct _VsxParameterList

  *List of parameter.*
- struct _VsxStatusItem

  *Declaration of status item.*
- struct _VsxStatusItemList

  *List of status items.*

## Macros

- #define LOCAL_INT64_T int64_t
- #define LOCAL_UINT64_T uint64_t

## Typedefs

- typedef enum _vsxStrategy VsxStrategy

  *The strategy which containers are removed when max number of items is reached.*
- typedef enum _vsxStatusCode VsxStatusCode

  *The status code for function calls.*
- typedef enum _vsxSerialConnectionType VsxSerialConnectionType

  *Defintion of serial connection type.*
- typedef struct _VsxSystemHandle VsxSystemHandle

  *Structure to use for sensor instance.*
- typedef enum _vsxDisconnectEvent VsxDisconnectEvent

  *status code of dinconnect event*
- typedef void(∗ vsx_OnDisconnect) (int handle, const char ∗ipAddress, VsxDisconnectEvent disconnectEvent, const char ∗description)

  *Callback definition for disconnect event.*
- typedef enum _vsxSessionTypes VsxSessionTypes

  *Status type of session message.*
- typedef void(∗ vsx_OnSessionMessageReceived) (int handle, VsxSessionTypes sessionType, int timeout)

  *Callback defition for session message received.*
- typedef struct _VsxDataContainerHandle VsxDataContainerHandle

  *Structure to use for a data container instance.*
- typedef enum _vsxImageData2Format VsxImageData2Format

  *Defintion of multiple image data formats.*
- typedef struct _VsxImage VsxImage

  *Declaration of image data.*
- typedef enum _vsxLineDataFormat VsxLineDataFormat

  *Defines the components, that could be part of line data.*
- typedef struct _VsxLineCoordinate VsxLineCoordinate

  *Declare coordinate point of line.*
- typedef struct _VsxLineData VsxLineData

  *Declare a line package.*
- typedef struct _VsxDisparityDescriptor2 VsxDisparityDescriptor2

  *Disparity descriptor to calculate 3D data from disparity map.*
- typedef struct _VsxTransformation VsxTransformation

  *Transformation containg translation and quaternion.*
- typedef struct _VsxCaptureInformation VsxCaptureInformation

  *Contains information about image capture.*
- typedef struct _VsxOlr2CaptureInformation VsxOlr2CaptureInformation

  *Contains information about image capture.*
- typedef struct _VsxOlr2ModbusData VsxOlr2ModbusData

  *Contains information about image capture.*
- typedef struct _VsxTagList VsxTagList

  *List of all possible tags inside a dynamic container.*
- typedef struct _VsxDevice VsxDevice

  *Declare device informations.*
- typedef struct _VsxDeviceList VsxDeviceList

  *List of devices.*
- typedef enum _vsxParameterValueType VsxParameterValueType

  *Define value type of parameter.*
- typedef struct _VsxParameterEnumItem VsxParameterEnumItem

  *Single item of a parameter enum.*

- typedef struct _VsxParameter VsxParameter

  *Declares parameter.*
- typedef struct _VsxParameterList VsxParameterList

  *List of parameter.*
- typedef enum _vsxStatusItemValueType VsxStatusItemValueType

  *Value types of status item.*
- typedef enum _vsxDeviceStatusScope VsxDeviceStatusScope

  *Scope of device status.*
- typedef struct _VsxStatusItem VsxStatusItem

  *Declaration of status item.*
- typedef struct _VsxStatusItemList VsxStatusItemList

  *List of status items.*
- typedef void(∗ vsx_OnDeviceStatusReceived) (int handle, VsxDeviceStatusScope deviceStatusScope, const VsxStatusItemList ∗statusItemListData)

  *Definition of callback function.*

### Enumerations

- enum _vsxStrategy { VSX_STRATEGY_DROP_OLDEST = 0 , VSX_STRATEGY_DROP_WRITE = 1 }

  *The strategy which containers are removed when max number of items is reached.*
- enum _vsxStatusCode {
  VSX_STATUS_SUCCESS = 0 , VSX_STATUS_ERROR_DRIVER_INIT = -0x1 , VSX_STATUS_ERROR_DRIVER_TIMEOUT = -0x2 , VSX_STATUS_ERROR_DRIVER_SAVE_FILE = -0x3 ,
  VSX_STATUS_ERROR_DRIVER_DATA = -0x4 , VSX_STATUS_ERROR_DRIVER_CONNECTION = -0x5 , VSX_STATUS_ERROR_DRIVER_INVALID_DATA = -0x6 , VSX_STATUS_ERROR_DRIVER_DEVICE = -0x7 ,
  VSX_STATUS_ERROR_DRIVER_LOAD_FILE = -0x8 , VSX_STATUS_ERROR_SESSION = -0x9 , VSX_STATUS_ERROR_STRING = -0x0A , VSX_STATUS_ERROR_VERSION = -0x0B ,
  VSX_STATUS_ERROR_DRIVER_GENERAL = -0x1000 , VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_SYSTEM = -0x8001 , VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_ZERO = -0x8002 , VSX_STATUS_ERROR_VSX_SYSTEM
  = -0x8003 ,
  VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_AVAILABLE = -0x8004 , VSX_STATUS_ERROR_MISSING_IP_ADDR
  = -0x8005 , VSX_STATUS_ERROR_MISSING_SERIALPORT_DECLARATION = -0x8006 , VSX_STATUS_ERROR_VSX_SYST
  = -0x8007 ,
  VSX_STATUS_ERROR_CONFIGURATION_ID_ZERO = -0x8008 , VSX_STATUS_ERROR_PARAMETER_ID_ZERO = -0x8009 , VSX_STATUS_ERROR_VALUE_ZERO = -0x800A , VSX_STATUS_ERROR_COMMAND_ZERO = -0x800B ,
  VSX_STATUS_ERROR_INPUT_VALUE_ZERO = -0x800C , VSX_STATUS_ERROR_OUTPUT_VALUE_POINTER_ZERO = -0x800D , VSX_STATUS_ERROR_OUTPUT_VALUE_NOT_ZERO = -0x800E , VSX_STATUS_ERROR_VALUE_POINTER_Z
  = -0x800F ,
  VSX_STATUS_ERROR_VALUE_NOT_ZERO = -0x8010 , VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX_SYSTEM = -0x8011 , VSX_STATUS_ERROR_XML_COMMAND_ZERO = -0x8012 , VSX_STATUS_ERROR_FILENAME_ZERO = -0x8013 ,
  VSX_STATUS_ERROR_STRING_POINTER_ZERO = -0x8014 , VSX_STATUS_ERROR_STRING_ZERO = -0x8015 , VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_POINTER_ZERO = -0x8016 , VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_DATA_CONTAINER = -0x8017 ,
  VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_NOT_ZERO = -0x8018 , VSX_STATUS_ERROR_VSX_DATA_CO
  = -0x8019 , VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_NOT_AVAILABLE = -0x801A , VSX_STATUS_ERROR_IMAGE_TAG_ZERO = -0x801B ,
  VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX_DATA_CONTAINER = -0x801C , VSX_STATUS_ERROR_UNABLE_TO_FIN
  = -0x801D , VSX_STATUS_ERROR_UNABLE_TO_FIND_IMAGE_TAG_TO_DATA_FORMAT = -0x801E , VSX_STATUS_ERROR_POINT_Z_ID_ZERO = -0x801F ,
  VSX_STATUS_ERROR_POINT_Y_ID_ZERO = -0x8020 , VSX_STATUS_ERROR_POINT_X_ID_ZERO = -0x8021 , VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_X_ID_IN_DATA_CONTAINER = -0x8022 , VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Y_ID_IN_DATA_CONTAINER = -0x8023 ,

VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_IN_DATA_CONTAINER = -0x8024 , VSX_STATUS_ERROR_UNA
= -0x8025 , VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Y_ID_TO_DATA_FORMAT = -0x8026 ,
VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_TO_DATA_FORMAT = -0x8027 ,
VSX_STATUS_ERROR_LOG_POINTER_ZERO = -0x8028 , VSX_STATUS_ERROR_LOG_NOT_ZERO = -0x8029 , VSX_STATUS_ERROR_RESULT_NOT_ZERO = -0x802A , VSX_STATUS_ERROR_RESULT_POINTER_ZERO
= -0x802B ,
VSX_STATUS_ERROR_UNABLE_TO_FIND_RESULT_ID_IN_DATA_CONTAINER = -0x802C , VSX_STATUS_ERROR_UNAB
= -0x802D , VSX_STATUS_ERROR_VERSION_POINTER_ZERO = -0x802E , VSX_STATUS_ERROR_VERSION_NOT_ZER
= -0x802F ,
VSX_STATUS_ERROR_VSX_IMAGE_POINTER_ZERO = -0x8030 , VSX_STATUS_ERROR_VSX_IMAGE_NOT_ZERO
= -0x8031 , VSX_STATUS_ERROR_UNDEFINED_STRATEGY_VALUE = -0x8032 , VSX_STATUS_ERROR_UNDEFINED_CO
= -0x8033 ,
VSX_STATUS_ERROR_XPATH_ZERO = -0x8034 , VSX_STATUS_ERROR_INVALID_DATA_FORMAT = -0x8035 , VSX_STATUS_ERROR_NO_ELEMENT_FOUND = -0x8036 , VSX_STATUS_ERROR_RESULT_TAG_ZERO
= -0x8037 ,
VSX_STATUS_ERROR_TAG_ZERO = -0x8038 , VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG_IN_DATA_CONTAINER
= -0x8039 , VSX_STATUS_ERROR_IP_ADDRESS_ZERO = -0x803A , VSX_STATUS_ERROR_NETWORK_MASK_ZERO
= -0x803B ,
VSX_STATUS_ERROR_GATEWAY_ZERO = -0x803C , VSX_STATUS_ERROR_EXCEPTION_THROWN = -0x803D , VSX_STATUS_ERROR_VSX_DEVICE_POINTER_ZERO = -0x803E , VSX_STATUS_ERROR_VSX_DEVICE_NOT_
= -0x803F ,
VSX_STATUS_ERROR_VSX_IMAGE_ZERO = -0x8040 , VSX_STATUS_ERROR_VSX_DEVICE_ZERO = -0x8041 , VSX_STATUS_ERROR_VSX_DEVICE_LIST_POINTER_ZERO = -0x8042 , VSX_STATUS_ERROR_VSX_DEVICE_L
= -0x8043 ,
VSX_STATUS_ERROR_VSX_TAG_LIST_ZERO = -0x8044 , VSX_STATUS_ERROR_VSX_TAG_LIST_POINTER_ZERO
= -0x8045 , VSX_STATUS_ERROR_VSX_TAG_LIST_NOT_ZERO = -0x8046 , VSX_STATUS_ERROR_VSX_PARAMETER_LI
= -0x8047 ,
VSX_STATUS_ERROR_VSX_PARAMETER_LIST_ZERO = -0x8048 , VSX_STATUS_ERROR_VSX_PARAMETER_NOT_ZER
= -0x8049 , VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_POINTER_ZERO = -0x804A , VSX_STATUS_ERROR_VSX_S
= -0x804B ,
VSX_STATUS_ERROR_VSX_STATUS_ITEM_NOT_ZERO = -0x0804C , VSX_STATUS_ERROR_ERROR_TEXT_POINTER_
= -0x0804D , VSX_STATUS_ERROR_ERROR_TEXT_NOT_ZERO = -0x0804E , VSX_STATUS_ERROR_ON_DISCONNECT_
= -0x0804F ,
VSX_STATUS_ERROR_MAC_ADDRESS_ZERO = -0x08050 , VSX_STATUS_ERROR_VSX_CACHED_CONTAINER_NOT_F
= -0x08051 , VSX_STATUS_ERROR_VSX_PARAMETER_LIST_NOT_ZERO = -0x08052 , VSX_STATUS_ERROR_VSX_PARA
= -0x08053 ,
VSX_STATUS_ERROR_VSX_PARAMETER_ZERO = -0x08054 , VSX_STATUS_ERROR_VSX_LINE_DATA_POINTER_ZERO
= -0x08064 , VSX_STATUS_ERROR_LINE_DATA_TAG_ZERO = -0x08065 , VSX_STATUS_ERROR_UNABLE_TO_FIND_LIN
= -0x08066 ,
VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_TAG_TO_DATA_FORMAT = -0x08067 , VSX_STATUS_ERROR_VSX_LIN
= -0x08068 , VSX_STATUS_ERROR_VSX_LINE_DATA_ZERO = -0x08069 , VSX_STATUS_ERROR_MISSING_LOGIN_PASS
= -0x0806A ,
VSX_STATUS_ERROR_MISSING_LOGIN_USERNAME = -0x0806B , VSX_STATUS_ERROR_ON_SESSION_MESSAGE_RE
= -0x0806C , VSX_STATUS_ERROR_VSX_PARAMETER_IN_POINTER_ZERO = -0x0806D , VSX_STATUS_ERROR_VSX_P
= -0x0806E ,
VSX_STATUS_ERROR_VSX_VALUE_POINTER_ZERO = -0x0806F , VSX_STATUS_ERROR_DATA_POINTER_ZERO
= -0x08070 , VSX_STATUS_ERROR_UNABLE_TO_FIND_MESSAGE_IN_DATA_CONTAINER = -0x08071 ,
VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_NOT_ZERO = -0x08072 ,
VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_ZERO = -0x08073 , VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG
= -0x08074 , VSX_STATUS_ERROR_INCORRECT_MESSAGE_FROM_TAG = -0x08075 }

*The status code for function calls.*

- enum _vsxSerialConnectionType {
VSX_SERIAL_CONNECTION_TYPE_USB_SSI = 0 , VSX_SERIAL_CONNECTION_TYPE_PROFIBUS = 1
, VSX_SERIAL_CONNECTION_TYPE_PROFINET = 2 , VSX_SERIAL_CONNECTION_TYPE_ETHERNET_IP
= 3 ,
VSX_SERIAL_CONNECTION_TYPE_RS485 = 4 , VSX_SERIAL_CONNECTION_TYPE_CANOPEN = 5 }

  *Defintion of serial connection type.*

- enum _vsxDisconnectEvent { VSX_DISCONNECT_EVENT_REMOTE_HOST_CONNECTION_CLOSED = 0

, VSX_DISCONNECT_EVENT_DISCONNECT_CALLED = 1 , VSX_DISCONNECT_EVENT_CONNECTION_ERROR = 2 }

  *status code of dinconnect event*

- enum _vsxSessionTypes { VSX_SESSION_TYPES_LOGIN_REQUIRED = 0 , VSX_SESSION_TYPES_INITIAL_PASSWORD_REQUIRED = 1 , VSX_SESSION_TYPES_LOGIN = 2 , VSX_SESSION_TYPES_LOGIN_REPLY = 3 , VSX_SESSION_TYPES_SET_PASSWORD = 4 , VSX_SESSION_TYPES_SET_PASSWORD_REPLY = 5 , VSX_SESSION_TYPES_TIMEOUT_ANNOUNCEMENT = 6 , VSX_SESSION_TYPES_TIMEOUT = 7 , VSX_SESSION_TYPES_LOGOUT = 8 , VSX_SESSION_TYPES_LOGOUT_REPLY = 9 , VSX_SESSION_TYPES_UNKNOWN = 10 }

  *Status type of session message.*
- enum _vsxImageData2Format { VSX_IMAGE_DATA2_FORMAT_MONO8 = 17301505 , VSX_IMAGE_DATA2_FORMAT_CONFIDENCE8 = 17301702 , VSX_IMAGE_DATA2_FORMAT_MONO12 = 17825797 , VSX_IMAGE_DATA2_FORMAT_MONO16 = 17825799 , VSX_IMAGE_DATA2_FORMAT_COORD3D_A16 = 17825974 , VSX_IMAGE_DATA2_FORMAT_COORD3D_B16 = 17825975 , VSX_IMAGE_DATA2_FORMAT_COORD3D_C16 = 17825976 , VSX_IMAGE_DATA2_FORMAT_COORD3D_A32 = 18874557 , VSX_IMAGE_DATA2_FORMAT_COORD3D_B32F = 18874558 , VSX_IMAGE_DATA2_FORMAT_COORD3D_C32F = 18874559 }

  *Defintion of multiple image data formats.*
- enum _vsxLineDataFormat { VSX_LINE_DATA_FORMAT_C = 0x00 , VSX_LINE_DATA_FORMAT_X = 0x01 , VSX_LINE_DATA_FORMAT_Y = 0x02 , VSX_LINE_DATA_FORMAT_Z = 0x04 , VSX_LINE_DATA_FORMAT_Q = 0x08 , VSX_LINE_DATA_FORMAT_I = 0x10 , VSX_LINE_DATA_FORMAT_32BIT_MODE = 0x1000 }

  *Defines the components, that could be part of line data.*
- enum _vsxParameterValueType { VSX_PARAMETER_VALUE_TYPE_BOOL = 0 , VSX_PARAMETER_VALUE_TYPE_INT = 1 , VSX_PARAMETER_VALUE_TYP = 2 , VSX_PARAMETER_VALUE_TYPE_UINT = 3 , VSX_PARAMETER_VALUE_TYPE_INT16 = 4 , VSX_PARAMETER_VALUE_TYPE_FLOAT = 5 , VSX_PARAMETER_VALUE_TYPE_DOUBLE = 6 , VSX_PARAMETER_VALUE_TYPE_STRING = 7 , VSX_PARAMETER_VALUE_TYPE_HEXSTRING = 8 , VSX_PARAMETER_VALUE_TYPE_BASE64 = 9 , VSX_PARAMETER_VALUE_TYPE_ENUM = 10 , VSX_PARAMETER_VALUE_TYPE_IP = 11 , VSX_PARAMETER_VALUE_TYPE_RECTANGLE = 12 , VSX_PARAMETER_VALUE_TYPE_QUAD = 13 , VSX_PARAMETER_VALUE_TYPE_POINT = 14 , VSX_PARAMETER_VALUE_TYPE_UNKNOWN = 15 }

  *Define value type of parameter.*
- enum _vsxStatusItemValueType { VSX_STATUS_ITEM_VALUE_TYPE_BOOL = 0 , VSX_STATUS_ITEM_VALUE_TYPE_INT = 1 , VSX_STATUS_ITEM_VALUE_TYPE_LONG = 2 , VSX_STATUS_ITEM_VALUE_TYPE_UINT = 3 , VSX_STATUS_ITEM_VALUE_TYPE_INT16 = 4 , VSX_STATUS_ITEM_VALUE_TYPE_FLOAT = 5 , VSX_STATUS_ITEM_VALUE_TYPE_DOUBLE = 6 , VSX_STATUS_ITEM_VALUE_TYPE_STRING = 7 , VSX_STATUS_ITEM_VALUE_TYPE_HEXSTRING = 8 , VSX_STATUS_ITEM_VALUE_TYPE_BASE64 = 9 , VSX_STATUS_ITEM_VALUE_TYPE_ENUM = 10 , VSX_STATUS_ITEM_VALUE_TYPE_IP = 11 , VSX_STATUS_ITEM_VALUE_TYPE_RECTANGLE = 12 , VSX_STATUS_ITEM_VALUE_TYPE_QUAD = 13 , VSX_STATUS_ITEM_VALUE_TYPE_POINT = 14 , VSX_STATUS_ITEM_VALUE_TYPE_UNKNOWN = 15 }

  *Value types of status item.*
- enum _vsxDeviceStatusScope { VSX_DEVICE_STATUS_SCOPE_FULL = 0 , VSX_DEVICE_STATUS_SCOPE_MULTI = 1 }

  *Scope of device status.*

## Functions

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseString (const char ∗∗pString)

*Release memory of string allocated by a function before. Sets the pointer to zero.*

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLibraryVersion (const char ∗∗version)

  *Returns the actual library version. PLease free "version" parameter after usage with 'vsx_ReleaseString'.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetErrorText (int32_t error_code, const char ∗∗error_text)

  *Return the error text to a given error code. It also appends additional text from last error given.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitTcpSensor (VsxSystemHandle ∗∗pVsx, const char ∗ipAddress, const char ∗pluginName)

  *Initialize a new tcp based sensor.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitSerialSensor (VsxSystemHandle ∗∗pVsx, const char ∗serialPort, int32_t baudrate, const char ∗sensorType, VsxSerialConnectionType connectionType, const char ∗pluginName)

  *Inits an instance to communicate with a Vsx-Device via serial protocol.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseSensor (VsxSystemHandle ∗∗vsx)

  *Frees the given sensor.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectTcpDevice (VsxSystemHandle ∗vsx, const char ∗ipAddress)

  *Disconnects the device and reconnects with new connection settings.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectAndLoginTcpDevice (VsxSystemHandle ∗vsx, const char ∗ipAddress, const char ∗username, const char ∗password)

  *Disconnects the device and reconnects with new connection settings and login credentials.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectSerialDevice (VsxSystemHandle ∗vsx, const char ∗serialPort, int32_t baudrate, VsxSerialConnectionType connectionType)

  *Disconnects the device and reconnects with new connection settings.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Connect (VsxSystemHandle ∗vsx)

  *Connect with the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectEx (VsxSystemHandle ∗vsx, int32_t timeout_ms)

  *Connect with the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectAndLogin (VsxSystemHandle ∗vsx, const char ∗username, const char ∗password)

  *Connect with the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectExAndLogin (VsxSystemHandle ∗vsx, const char ∗username, const char ∗password, int32_t timeout_ms)

  *Connect with the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Login (VsxSystemHandle ∗vsx, const char ∗username, const char ∗password)

  *Login to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Logout (VsxSystemHandle ∗vsx)

  *Logout from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetPassword (VsxSystemHandle ∗vsx, const char ∗authorizationUsername, const char ∗authorizationPassword, const char ∗username, const char ∗password)

  *Set new password on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetConnected (VsxSystemHandle ∗vsx, int32_t ∗result)

  *Indicates current connection state with the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Disconnect (VsxSystemHandle ∗vsx)

  *Disconnect with the device.*

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDisconnect (VsxSystemHandle ∗vsx, vsx_OnDisconnect fptr)

  *Register callback for "vsx_OnDisconnect" callback. Only short execution times are allowed (do not block the function), just use to transfer data to your (main) thread.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDisconnect (VsxSystemHandle ∗vsx)

  *Function to deregister already existing callback function.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnSessionMessageReceived (VsxSystemHandle ∗vsx, vsx_OnSessionMessageReceived fptr)

  *Register callback for "vsx_OnSessionMessageReceived" callback. Only short execution times are allowed (do not block the function), just use to transfer data to your (main) thread.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnSessionMessageReceived (VsxSystemHandle ∗vsx)

  *Function to deregister already existing callback function.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendSessionKeepAlive (VsxSystemHandle ∗vsx)

  *Send session keep alive to sensor. Should be the reply from a timeout announcement message.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystem (VsxSystemHandle ∗vsx, const char ∗command, const char ∗inputValue, const char ∗∗outputValue, int32_t ∗status)

  *Sends a test system command to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystemEx (VsxSystemHandle ∗vsx, const char ∗command, const char ∗inputValue, const char ∗∗outputValue, int32_t ∗status, int32_t timeout_ms)

  *Sends a test system command to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetWaitTimeout (VsxSystemHandle ∗vsx, int32_t ∗timeout_ms)

  *Gets the time in ms, the driver waits for response from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetWaitTimeout (VsxSystemHandle ∗vsx, int32_t timeout_ms)

  *Sets the time in ms, the driver waits for response from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadData (VsxSystemHandle ∗vsx, const char ∗fileName)

  *Sends a data file (either image data or dynamic container data) to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendFirmware (VsxSystemHandle ∗vsx, const char ∗fileName)

  *Sends a firmware update file to the device. NOTE: not completely implemented yet, the file is send only to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendXmlDataMessage (VsxSystemHandle ∗vsx, const char ∗xmlCommand)

  *Sends a string to the device. NOTE: function does not wait for any device reply.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettings (VsxSystemHandle ∗vsx, const char ∗ipAddress, const char ∗networkMask, const char ∗gateway)

  *Sets the network settings of the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettingsViaUdp (const char ∗macAddress, const char ∗ipAddress, const char ∗networkMask, const char ∗gateway)

  *Sets the network settings of the device identified by the macAddress via UDP.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetDynamicContainerGrabber (VsxSystemHandle ∗vsx, int32_t bufferSize, VsxStrategy strategy)

  *Restarts the internal dynamic container grabber. Saving the items will be new initialized.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDataContainer (VsxSystemHandle ∗vsx, VsxDataContainerHandle ∗∗pDch, int32_t timeout_ms)

  *Gets the oldest saved item and removes it internally.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCachedContainer (VsxSystemHandle ∗vsx, VsxDataContainerHandle ∗∗pDch, int32_t position)

*Gets a cached dynamic container.*

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDataContainer (VsxDataContainerHandle ∗∗dch)

  *Release / Free data container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveData (VsxDataContainerHandle ∗dch, const char ∗tag, const char ∗fileName)

  *Saves a VsxMessage to the given filename.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Save3DPointCloudData (VsxDataContainerHandle ∗dch, const char ∗point_x_Id, const char ∗point_y_Id, const char ∗point_z_Id, const char ∗fileName)

  *Saves a 3D point cloud as pcd to the given filename.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetImage (VsxDataContainerHandle ∗dch, const char ∗tag, VsxImage ∗∗imageData)

  *Get image from a dynamic container, access via raw memory pointer.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseImage (VsxImage ∗∗p←Image)

  *Release / free image object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLine (VsxDataContainerHandle ∗dch, const char ∗tag, VsxLineData ∗∗data)

  *Get line data from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseLine (VsxLineData ∗∗p←LineData)

  *Release / free line data object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2 (VsxDataContainerHandle ∗dch, const char ∗tag, VsxDisparityDescriptor2 ∗∗data)

  *Get disparity descriptor from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDisparityDescriptor2 (VsxDisparityDescriptor2 ∗∗pData)

  *Release / free DisparityDescriptor2 object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTransformation (VsxDataContainerHandle ∗dch, const char ∗tag, VsxTransformation ∗∗data)

  *Get transformation from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTransformation (VsxTransformation ∗∗pData)

  *Release / free Transformation object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCaptureInformation (VsxDataContainerHandle ∗dch, const char ∗tag, VsxCaptureInformation ∗∗data)

  *Get capture information from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseCaptureInformation (VsxCaptureInformation ∗∗pData)

  *Release / free CaptureInformation object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2CaptureInformation (VsxDataContainerHandle ∗dch, const char ∗tag, VsxOlr2CaptureInformation ∗∗data)

  *Get olr2 capture information from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2CaptureInformation (VsxOlr2CaptureInformation ∗∗pData)

  *Release / free Olr2CaptureInformation object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2ModbusData (VsxDataContainerHandle ∗dch, const char ∗tag, VsxOlr2ModbusData ∗∗data)

  *Get modbus data for olr2 sensor from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2ModbusData (VsxOlr2ModbusData ∗∗pData)

  *Release / free Olr2ModbusData object.*

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTagList (VsxDataContainerHandle ∗dch, VsxTagList ∗∗tagList)

  *Returns all available tags from a dynamic container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTagList (VsxTagList ∗∗pTagList)

  *Relase / free release tag list object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingContainerFramesCounter (VsxSystemHandle ∗vsx, int32_t ∗result)

  *Gets the missing frame counter from image grabbing.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDynamicContainerQueueSize (VsxSystemHandle ∗vsx, int32_t ∗result)

  *Gets the current size of the dynamic container message queue.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetNumberOfCachedContainers (VsxSystemHandle ∗vsx, int32_t ∗result)

  *Gets the current number of cached container messages.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDeviceInformation (VsxSystemHandle ∗vsx, VsxDevice ∗∗deviceData)

  *Returns a device object with network information about the current device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDevice (VsxDevice ∗∗pDevice)

  *Release / Free existing handle to device object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetUdpDeviceList (VsxDeviceList ∗∗deviceListData)

  *Searches for all devices in a subnet via udp and returns a list with all devices found.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDeviceList (VsxDeviceList ∗∗pDeviceList)

  *Release / Free existing handle to device list object.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetLogMessageGrabber (VsxSystemHandle ∗vsx, int32_t bufferSize, int32_t typeMask, VsxStrategy strategy)

  *Starts the internal log message grabber. Saving the items will be new initialized.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessage (VsxSystemHandle ∗vsx, const char ∗∗log, int32_t timeout_ms)

  *Gets the oldest saved item and removes it internally.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessageQueueSize (VsxSystemHandle ∗vsx, int32_t ∗result)

  *Gets the current size of the log message queue.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingLogMessagesCounter (VsxSystemHandle ∗vsx, int32_t ∗result)

  *Gets the missing log messages counter for log message grabbing.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValue (VsxSystemHandle ∗vsx, uint32_t settingsVersion, const char ∗configurationId, uint32_t configurationVersion, const char ∗parameterId, const char ∗value)

  *Sets the parameter to a value on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueDouble (VsxSystemHandle ∗vsx, uint32_t settingsVersion, const char ∗configurationId, uint32_t configurationVersion, const char ∗parameterId, double value)

  *Sets the parameter to a value on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueInt32 (VsxSystemHandle ∗vsx, uint32_t settingsVersion, const char ∗configurationId, uint32_t configurationVersion, const char ∗parameterId, int32_t value)

  *Sets the parameter to a value on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValue (VsxSystemHandle ∗vsx, uint32_t settingsVersion, const char ∗configurationId, uint32_t configurationVersion, const char ∗parameterId, const char ∗∗value)

*Returns the current value of the given parameter from device.*

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueDouble (VsxSystemHandle ∗vsx, uint32_t settingsVersion, const char ∗configurationId, uint32_t configurationVersion, const char ∗parameterId, double ∗value)

  *Returns the current value of the given parameter from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueInt32 (VsxSystemHandle ∗vsx, uint32_t settingsVersion, const char ∗configurationId, uint32_t configurationVersion, const char ∗parameterId, int32_t ∗value)

  *Returns the current value of the given parameter from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadDefaultParameterSetOnDevice (VsxSystemHandle ∗vsx)

  *Resets the devices parameters to factory settings and returns a list of the complete parameter set of the device including current values.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadParameterSetOnDevice (VsxSystemHandle ∗vsx)

  *Loads the parameter set saved on device and returns a list of the complete parameter set of the device including current values.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveParameterSetOnDevice (VsxSystemHandle ∗vsx)

  *Saves the current parameter set on device. Parameter values will be loaded when device starts.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterSet (VsxSystemHandle ∗vsx, const char ∗fileName)

  *Uploads a parameter file to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DownloadParameterSet (VsxSystemHandle ∗vsx, const char ∗fileName)

  *Save the current parameter set to a file.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetParameterList (VsxSystemHandle ∗vsx, VsxParameterList ∗∗parameterListData)

  *Returns a list of the complete parameter set of the device including their current values. The list shows the current state of the parameters.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterList (VsxSystemHandle ∗vsx, VsxParameterList ∗parameterListData)

  *Uploads a parameter list to the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterString (VsxSystemHandle ∗vsx, const VsxParameter ∗parameter, const char ∗value)

  *Sets the parameter to a value on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble (VsxSystemHandle ∗vsx, const VsxParameter ∗parameter, double value)

  *Sets the parameter to a value on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterInt32 (VsxSystemHandle ∗vsx, const VsxParameter ∗parameter, int32_t value)

  *Sets the parameter to a value on the device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameter (VsxSystemHandle ∗vsx, const VsxParameter ∗parameterIn, const VsxParameter ∗∗parameterOut)

  *Returns the current value of the given parameter from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameter (const VsxParameter ∗∗pParameter)

  *Release parameter handle.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameterList (VsxParameterList ∗∗pParameterList)

  *Release parameter list handle.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultXml (VsxDataContainerHandle ∗dch, const char ∗resultId, const char ∗∗result)

  *Returns the complete xml response from an result inside data container.*

- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementString (VsxDataContainerHandle ∗dch, const char ∗resultId, const char ∗xPath, const char ∗∗result)

  *Return certain value from a result inside data container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32 (VsxDataContainerHandle ∗dch, const char ∗resultId, const char ∗xPath, int32_t ∗result)

  *Return certain value from a result inside data container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt64 (VsxDataContainerHandle ∗dch, const char ∗resultId, const char ∗xPath, LOCAL_INT64_T ∗result)

  *Return certain value from a result inside data container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementDouble (VsxDataContainerHandle ∗dch, const char ∗resultId, const char ∗xPath, double ∗result)

  *Return certain value from a result inside data container.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetAllDeviceStatusData (VsxSystemHandle ∗vsx, VsxStatusItemList ∗∗statusItemListData)

  *Get the full status data set from device.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseStatusItemList (VsxStatusItemList ∗∗pStatusItemList)

  *Release / Free "VsxStatusItemList" handle.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDeviceStatusReceived (VsxSystemHandle ∗vsx, vsx_OnDeviceStatusReceived fptr)

  *Register callback for "vsx_OnDeviceStatusReceived" callback. Only short execution times are allowed (do not block the function), just use to transfer data to your (main) thread.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDeviceStatusReceived (VsxSystemHandle ∗vsx)

  *Function to deregister already existing callback function.*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SubscribeToDeviceStatusData (VsxSystemHandle ∗vsx)

  *Subscribe status data from sensor to the client. This will send periodically or in case of a problem status data to the client. This need a registered callback for "vsx_OnDeviceStatusReceived".*
- DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UnsubscribeToDeviceStatusData (VsxSystemHandle ∗vsx)

  *Unsubscribe status data from sensor.*

### 9.8.1 Macro Definition Documentation

#### LOCAL_INT64_T

```
#define LOCAL_INT64_T int64_t
```

Helper function to allow (really) old compiler running code The 64bit support for the old compiler is splitted into two variables. Signed values will be splitted unsigned values. Watch out, when using negative values.

#### LOCAL_UINT64_T

```
#define LOCAL_UINT64_T uint64_t
```

### 9.8.2 Typedef Documentation

#### VsxStrategy

```
typedef enum _vsxStrategy VsxStrategy
```

The strategy which containers are removed when max number of items is reached.

### VsxStatusCode

`typedef enum _vsxStatusCode VsxStatusCode`

The status code for function calls.

### VsxSerialConnectionType

`typedef enum _vsxSerialConnectionType VsxSerialConnectionType`

Defintion of serial connection type.

### VsxSystemHandle

`typedef struct _VsxSystemHandle VsxSystemHandle`

Structure to use for sensor instance.

### VsxDisconnectEvent

`typedef enum _vsxDisconnectEvent VsxDisconnectEvent`

status code of dinconnect event

### vsx_OnDisconnect

```
typedef void(* vsx_OnDisconnect) (int handle, const char *ipAddress, VsxDisconnectEvent disconnect↩
Event, const char *description)
```

Callback definition for disconnect event.

### VsxSessionTypes

`typedef enum _vsxSessionTypes VsxSessionTypes`

Status type of session message.

### vsx_OnSessionMessageReceived

```
typedef void(* vsx_OnSessionMessageReceived) (int handle, VsxSessionTypes sessionType, int
timeout)
```

Callback defition for session message received.

### VsxDataContainerHandle

`typedef struct _VsxDataContainerHandle VsxDataContainerHandle`

Structure to use for a data container instance.

### VsxImageData2Format

`typedef enum _vsxImageData2Format VsxImageData2Format`

Defintion of multiple image data formats.

### VsxImage

`typedef struct _VsxImage VsxImage`

Declaration of image data.

### VsxLineDataFormat

`typedef enum _vsxLineDataFormat VsxLineDataFormat`

Defines the components, that could be part of line data.

### VsxLineCoordinate

`typedef struct _VsxLineCoordinate VsxLineCoordinate`

Declare coordinate point of line.

### VsxLineData

`typedef struct _VsxLineData VsxLineData`

Declare a line package.

### VsxDisparityDescriptor2

`typedef struct _VsxDisparityDescriptor2 VsxDisparityDescriptor2`

Disparity descriptor to calculate 3D data from disparity map.

### VsxTransformation

`typedef struct _VsxTransformation VsxTransformation`

Transformation containg translation and quaternion.

### VsxCaptureInformation

`typedef struct _VsxCaptureInformation VsxCaptureInformation`

Contains information about image capture.

### VsxOlr2CaptureInformation

`typedef struct _VsxOlr2CaptureInformation VsxOlr2CaptureInformation`

Contains information about image capture.

### VsxOlr2ModbusData

`typedef struct _VsxOlr2ModbusData VsxOlr2ModbusData`

Contains information about image capture.

### VsxTagList

`typedef struct _VsxTagList VsxTagList`

List of all possible tags inside a dynamic container.

### VsxDevice

`typedef struct _VsxDevice VsxDevice`

Declare device informations.

### VsxDeviceList

`typedef struct _VsxDeviceList VsxDeviceList`

List of devices.

### VsxParameterValueType

`typedef enum _vsxParameterValueType VsxParameterValueType`

Define value type of parameter.

### VsxParameterEnumItem

`typedef struct _VsxParameterEnumItem VsxParameterEnumItem`

Single item of a parameter enum.

#### VsxParameter

```
typedef struct _VsxParameter VsxParameter
```

Declares parameter.

#### VsxParameterList

```
typedef struct _VsxParameterList VsxParameterList
```

List of parameter.

#### VsxStatusItemValueType

```
typedef enum _vsxStatusItemValueType VsxStatusItemValueType
```

Value types of status item.

#### VsxDeviceStatusScope

```
typedef enum _vsxDeviceStatusScope VsxDeviceStatusScope
```

Scope of device status.

#### VsxStatusItem

```
typedef struct _VsxStatusItem VsxStatusItem
```

Declaration of status item.

#### VsxStatusItemList

```
typedef struct _VsxStatusItemList VsxStatusItemList
```

List of status items.

#### vsx_OnDeviceStatusReceived

```
typedef void(* vsx_OnDeviceStatusReceived) (int handle, VsxDeviceStatusScope deviceStatus↩
Scope, const VsxStatusItemList *statusItemListData)
```

Definition of callback function.

### 9.8.3 Enumeration Type Documentation

#### _vsxStrategy

```
enum _vsxStrategy
```

The strategy which containers are removed when max number of items is reached.

**Enumerator**

| | |
|---|---|
| VSX_STRATEGY_DROP_OLDEST | Discards the oldest saved item if max number of items is reached. |
| VSX_STRATEGY_DROP_WRITE | Discards the current item if max number of items is reached. |

## _vsxStatusCode

`enum _vsxStatusCode`

The status code for function calls.

**Enumerator**

| | |
|---|---|
| VSX_STATUS_SUCCESS | Success of function call. This one should be used for checking of an error. |
| VSX_STATUS_ERROR_DRIVER_INIT | |
| VSX_STATUS_ERROR_DRIVER_TIMEOUT | |
| VSX_STATUS_ERROR_DRIVER_SAVE_FILE | |
| VSX_STATUS_ERROR_DRIVER_DATA | |
| VSX_STATUS_ERROR_DRIVER_CONNECTION | |
| VSX_STATUS_ERROR_DRIVER_INVALID_DATA | |
| VSX_STATUS_ERROR_DRIVER_DEVICE | |
| VSX_STATUS_ERROR_DRIVER_LOAD_FILE | |
| VSX_STATUS_ERROR_SESSION | |
| VSX_STATUS_ERROR_STRING | |
| VSX_STATUS_ERROR_VERSION | |
| VSX_STATUS_ERROR_DRIVER_GENERAL | |
| VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_SYSTEM | |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_ZERO | |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_AVAILABLE | |
| VSX_STATUS_ERROR_MISSING_IP_ADDRESS_DECLARATION | |
| VSX_STATUS_ERROR_MISSING_SERIALPORT_DECLARATION | |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_POINTER_ZERO | |
| VSX_STATUS_ERROR_CONFIGURATION_ID_ZERO | |
| VSX_STATUS_ERROR_PARAMETER_ID_ZERO | |
| VSX_STATUS_ERROR_VALUE_ZERO | |
| VSX_STATUS_ERROR_COMMAND_ZERO | |
| VSX_STATUS_ERROR_INPUT_VALUE_ZERO | |
| VSX_STATUS_ERROR_OUTPUT_VALUE_POINTER_ZERO | |
| VSX_STATUS_ERROR_OUTPUT_VALUE_NOT_ZERO | |
| VSX_STATUS_ERROR_VALUE_POINTER_ZERO | |

**Enumerator**

| | |
|---|---|
| VSX_STATUS_ERROR_VALUE_NOT_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX↩ _SYSTEM | |
| VSX_STATUS_ERROR_XML_COMMAND_ZERO | |
| VSX_STATUS_ERROR_FILENAME_ZERO | |
| VSX_STATUS_ERROR_STRING_POINTER_ZERO | |
| VSX_STATUS_ERROR_STRING_ZERO | |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↩ _HANDLE_POINTER_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE↩ _VSX_DATA_CONTAINER | |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↩ _HANDLE_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↩ _HANDLE_ZERO | |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↩ _HANDLE_NOT_AVAILABLE | |
| VSX_STATUS_ERROR_IMAGE_TAG_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX↩ _DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ IMAGE_ID_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ IMAGE_TAG_TO_DATA_FORMAT | |
| VSX_STATUS_ERROR_POINT_Z_ID_ZERO | |
| VSX_STATUS_ERROR_POINT_Y_ID_ZERO | |
| VSX_STATUS_ERROR_POINT_X_ID_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ POINT_X_ID_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ POINT_Y_ID_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ POINT_Z_ID_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ POINT_X_ID_TO_DATA_FORMAT | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ POINT_Y_ID_TO_DATA_FORMAT | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ POINT_Z_ID_TO_DATA_FORMAT | |
| VSX_STATUS_ERROR_LOG_POINTER_ZERO | |
| VSX_STATUS_ERROR_LOG_NOT_ZERO | |
| VSX_STATUS_ERROR_RESULT_NOT_ZERO | |
| VSX_STATUS_ERROR_RESULT_POINTER_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ RESULT_ID_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↩ RESULT_ID_TO_DATA_FORMAT | |
| VSX_STATUS_ERROR_VERSION_POINTER_ZERO | |
| VSX_STATUS_ERROR_VERSION_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_IMAGE_POINTER_↩ ZERO | |

**Enumerator**

| | |
|---|---|
| VSX_STATUS_ERROR_VSX_IMAGE_NOT_ZERO | |
| VSX_STATUS_ERROR_UNDEFINED_STRATEGY_VALUE | |
| VSX_STATUS_ERROR_UNDEFINED_CONNECTION_TYPE_VALUE | |
| VSX_STATUS_ERROR_XPATH_ZERO | |
| VSX_STATUS_ERROR_INVALID_DATA_FORMAT | |
| VSX_STATUS_ERROR_NO_ELEMENT_FOUND | |
| VSX_STATUS_ERROR_RESULT_TAG_ZERO | |
| VSX_STATUS_ERROR_TAG_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_IP_ADDRESS_ZERO | |
| VSX_STATUS_ERROR_NETWORK_MASK_ZERO | |
| VSX_STATUS_ERROR_GATEWAY_ZERO | |
| VSX_STATUS_ERROR_EXCEPTION_THROWN | |
| VSX_STATUS_ERROR_VSX_DEVICE_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_DEVICE_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_IMAGE_ZERO | |
| VSX_STATUS_ERROR_VSX_DEVICE_ZERO | |
| VSX_STATUS_ERROR_VSX_DEVICE_LIST_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_DEVICE_LIST_ZERO | |
| VSX_STATUS_ERROR_VSX_TAG_LIST_ZERO | |
| VSX_STATUS_ERROR_VSX_TAG_LIST_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_TAG_LIST_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_PARAMETER_LIST_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_PARAMETER_LIST_ZERO | |
| VSX_STATUS_ERROR_VSX_PARAMETER_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_ZERO | |
| VSX_STATUS_ERROR_VSX_STATUS_ITEM_NOT_ZERO | |
| VSX_STATUS_ERROR_ERROR_TEXT_POINTER_ZERO | |
| VSX_STATUS_ERROR_ERROR_TEXT_NOT_ZERO | |
| VSX_STATUS_ERROR_ON_DISCONNECT_CALLBACK_ZERO | |
| VSX_STATUS_ERROR_MAC_ADDRESS_ZERO | |
| VSX_STATUS_ERROR_VSX_CACHED_CONTAINER_NOT_FOUND | |
| VSX_STATUS_ERROR_VSX_PARAMETER_LIST_NOT_ZERO | |

**Enumerator**

| | |
|---|---|
| VSX_STATUS_ERROR_VSX_PARAMETER_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_PARAMETER_ZERO | |
| VSX_STATUS_ERROR_VSX_LINE_DATA_POINTER_ZERO | |
| VSX_STATUS_ERROR_LINE_DATA_TAG_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_ID_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_TAG_TO_DATA_FORMAT | |
| VSX_STATUS_ERROR_VSX_LINE_NOT_ZERO | |
| VSX_STATUS_ERROR_VSX_LINE_DATA_ZERO | |
| VSX_STATUS_ERROR_MISSING_LOGIN_PASSWORD | |
| VSX_STATUS_ERROR_MISSING_LOGIN_USERNAME | |
| VSX_STATUS_ERROR_ON_SESSION_MESSAGE_RECEIVED_CALLBACK_ZERO | |
| VSX_STATUS_ERROR_VSX_PARAMETER_IN_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_PARAMETER_OUT_POINTER_ZERO | |
| VSX_STATUS_ERROR_VSX_VALUE_POINTER_ZERO | |
| VSX_STATUS_ERROR_DATA_POINTER_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_MESSAGE_IN_DATA_CONTAINER | |
| VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_NOT_ZERO | |
| VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_ZERO | |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG | |
| VSX_STATUS_ERROR_INCORRECT_MESSAGE_FROM_TAG | |

### _vsxSerialConnectionType

`enum _vsxSerialConnectionType`

Defintion of serial connection type.

**Enumerator**

| | |
|---|---|
| VSX_SERIAL_CONNECTION_TYPE_USB_SSI | |
| VSX_SERIAL_CONNECTION_TYPE_PROFIBUS | |
| VSX_SERIAL_CONNECTION_TYPE_PROFINET | |
| VSX_SERIAL_CONNECTION_TYPE_ETHERNET_IP | |
| VSX_SERIAL_CONNECTION_TYPE_RS485 | |
| VSX_SERIAL_CONNECTION_TYPE_CANOPEN | |

### _vsxDisconnectEvent

`enum _vsxDisconnectEvent`

status code of dinconnect event

**Enumerator**

| | |
|---|---|
| VSX_DISCONNECT_EVENT_REMOTE_HOST_CONNECTION_CLOSED | |
| VSX_DISCONNECT_EVENT_DISCONNECT_CALLED | |
| VSX_DISCONNECT_EVENT_CONNECTION_ERROR | |

### _vsxSessionTypes

`enum _vsxSessionTypes`

Status type of session message.

**Enumerator**

| | |
|---|---|
| VSX_SESSION_TYPES_LOGIN_REQUIRED | |
| VSX_SESSION_TYPES_INITIAL_PASSWORD_REQUIRED | |
| VSX_SESSION_TYPES_LOGIN | |
| VSX_SESSION_TYPES_LOGIN_REPLY | |
| VSX_SESSION_TYPES_SET_PASSWORD | |
| VSX_SESSION_TYPES_SET_PASSWORD_REPLY | |
| VSX_SESSION_TYPES_TIMEOUT_ANNOUNCEMENT | |
| VSX_SESSION_TYPES_TIMEOUT | |
| VSX_SESSION_TYPES_LOGOUT | |
| VSX_SESSION_TYPES_LOGOUT_REPLY | |
| VSX_SESSION_TYPES_UNKNOWN | |

### _vsxImageData2Format

`enum _vsxImageData2Format`

Defintion of multiple image data formats.

**Enumerator**

| | |
|---|---|
| VSX_IMAGE_DATA2_FORMAT_MONO8 | |
| VSX_IMAGE_DATA2_FORMAT_CONFIDENCE8 | |
| VSX_IMAGE_DATA2_FORMAT_MONO12 | |
| VSX_IMAGE_DATA2_FORMAT_MONO16 | |
| VSX_IMAGE_DATA2_FORMAT_COORD3D_A16 | |
| VSX_IMAGE_DATA2_FORMAT_COORD3D_B16 | |
| VSX_IMAGE_DATA2_FORMAT_COORD3D_C16 | |
| VSX_IMAGE_DATA2_FORMAT_COORD3D_A32F | |
| VSX_IMAGE_DATA2_FORMAT_COORD3D_B32F | |
| VSX_IMAGE_DATA2_FORMAT_COORD3D_C32F | |

### _vsxLineDataFormat

`enum _vsxLineDataFormat`

Defines the components, that could be part of line data.

**Enumerator**

| | |
|---|---|
| VSX_LINE_DATA_FORMAT_C | |
| VSX_LINE_DATA_FORMAT_X | |
| VSX_LINE_DATA_FORMAT_Y | |
| VSX_LINE_DATA_FORMAT_Z | |
| VSX_LINE_DATA_FORMAT_Q | |
| VSX_LINE_DATA_FORMAT_I | |
| VSX_LINE_DATA_FORMAT_32BIT_MODE | |

### _vsxParameterValueType

`enum _vsxParameterValueType`

Define value type of parameter.

**Enumerator**

| | |
|---|---|
| VSX_PARAMETER_VALUE_TYPE_BOOL | Result in 'valueInt'. |
| VSX_PARAMETER_VALUE_TYPE_INT | Result in 'valueInt'. |
| VSX_PARAMETER_VALUE_TYPE_LONG | Result in 'valueInt'. |
| VSX_PARAMETER_VALUE_TYPE_UINT | Result in 'valueInt'. |
| VSX_PARAMETER_VALUE_TYPE_INT16 | Result in 'valueInt'. |
| VSX_PARAMETER_VALUE_TYPE_FLOAT | Result in 'valueDouble'. |
| VSX_PARAMETER_VALUE_TYPE_DOUBLE | Result in 'valueDouble'. |
| VSX_PARAMETER_VALUE_TYPE_STRING | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_HEXSTRING | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_BASE64 | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_ENUM | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_IP | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_RECTANGLE | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_QUAD | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_POINT | Result in 'valueString'. |
| VSX_PARAMETER_VALUE_TYPE_UNKNOWN | |

### _vsxStatusItemValueType

`enum _vsxStatusItemValueType`

Value types of status item.

**Enumerator**

| | |
|---|---|
| VSX_STATUS_ITEM_VALUE_TYPE_BOOL | Result in 'valueInt'. |
| VSX_STATUS_ITEM_VALUE_TYPE_INT | Result in 'valueInt'. |
| VSX_STATUS_ITEM_VALUE_TYPE_LONG | Result in 'valueInt'. |
| VSX_STATUS_ITEM_VALUE_TYPE_UINT | Result in 'valueInt'. |
| VSX_STATUS_ITEM_VALUE_TYPE_INT16 | Result in 'valueInt'. |
| VSX_STATUS_ITEM_VALUE_TYPE_FLOAT | Result in 'valueDouble'. |
| VSX_STATUS_ITEM_VALUE_TYPE_DOUBLE | Result in 'valueDouble'. |
| VSX_STATUS_ITEM_VALUE_TYPE_STRING | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_HEXSTRING | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_BASE64 | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_ENUM | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_IP | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_RECTANGLE | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_QUAD | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_POINT | Result in 'valueString'. |
| VSX_STATUS_ITEM_VALUE_TYPE_UNKNOWN | |

### _vsxDeviceStatusScope

```
enum _vsxDeviceStatusScope
```

Scope of device status.

**Enumerator**

| | |
|---|---|
| VSX_DEVICE_STATUS_SCOPE_FULL | |
| VSX_DEVICE_STATUS_SCOPE_MULTI | |

## 9.8.4 Function Documentation

### vsx_ReleaseString()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseString (
            const char ** pString )
```

Release memory of string allocated by a function before. Sets the pointer to zero.

**Parameters**

| | |
|---|---|
| *pString* | Reference to string pointer |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetLibraryVersion()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLibraryVersion (
            const char ** version )
```

Returns the actual library version. PLease free "version" parameter after usage with 'vsx_ReleaseString'.

**Parameters**

| | |
|---|---|
| *version* | Reference to string pointer |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetErrorText()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetErrorText (
            int32_t error_code,
            const char ** error_text )
```

Return the error text to a given error code. It also appends additional text from last error given.

**Parameters**

| | |
|---|---|
| *error_code* | Input error code |
| *error_text* | Reference to string pointer |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_InitTcpSensor()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitTcpSensor (
            VsxSystemHandle ** pVsx,
            const char * ipAddress,
            const char * pluginName )
```

Initialize a new tcp based sensor.

**Parameters**

| | |
|---|---|
| *pVsx* | Reference to an empty vsx system handle |
| *ipAddress* | e.g. 192.168.2.4 |
| *pluginName* | Additional functionality for special sensors. E.g. 'SR3D_STEREO' to calculate 3D data from disparity map on target |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_InitSerialSensor()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitSerialSensor (
             VsxSystemHandle ** pVsx,
             const char * serialPort,
             int32_t baudrate,
             const char * sensorType,
             VsxSerialConnectionType connectionType,
             const char * pluginName )
```

Inits an instance to communicate with a Vsx-Device via serial protocol.

**Parameters**

| | |
|---|---|
| *pVsx* | New handle to sensor |
| *serialPort* | The comport of the device. |
| *baudrate* | The baudrate of the device. |
| *sensorType* | The sensor type of the device.> |
| *connectionType* | The connection type of the device. |
| *pluginName* | The type of the device. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseSensor()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseSensor (
             VsxSystemHandle ** vsx )
```

Frees the given sensor.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReConnectTcpDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectTcpDevice (
             VsxSystemHandle * vsx,
             const char * ipAddress )
```

Disconnects the device and reconnects with new connection settings.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *ipAddress* | The new IPAddress. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReConnectAndLoginTcpDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectAndLoginTcpDevice (
            VsxSystemHandle * vsx,
            const char * ipAddress,
            const char * username,
            const char * password )
```

Disconnects the device and reconnects with new connection settings and login credentials.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *ipAddress* | The new IPAddress. |
| *username* | username for login |
| *password* | password for login |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReConnectSerialDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectSerialDevice (
            VsxSystemHandle * vsx,
            const char * serialPort,
            int32_t baudrate,
            VsxSerialConnectionType connectionType )
```

Disconnects the device and reconnects with new connection settings.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *serialPort* | The new serial port. |
| *baudrate* | The new baudrate. |
| *connectionType* | The new connection type. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_Connect()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Connect (
            VsxSystemHandle * vsx )
```

Connect with the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ConnectEx()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectEx (
            VsxSystemHandle * vsx,
            int32_t timeout_ms )
```

Connect with the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *timeout_ms* | The timeout for a connection attempt |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ConnectAndLogin()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectAndLogin (
            VsxSystemHandle * vsx,
            const char * username,
            const char * password )
```

Connect with the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *username* | username for login |
| *password* | password for login |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ConnectExAndLogin()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectExAndLogin (
            VsxSystemHandle * vsx,
            const char * username,
            const char * password,
            int32_t timeout_ms )
```

Connect with the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *username* | username for login |
| *password* | password for login |
| *timeout_ms* | The timeout for a connection attempt |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_Login()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Login (
            VsxSystemHandle * vsx,
            const char * username,
            const char * password )
```

Login to the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *username* | username for login |
| *password* | password for login |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_Logout()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Logout (
            VsxSystemHandle * vsx )
```

Logout from device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetPassword()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetPassword (
            VsxSystemHandle ∗ vsx,
            const char ∗ authorizationUsername,
            const char ∗ authorizationPassword,
            const char ∗ username,
            const char ∗ password )
```

Set new password on the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *authorizationUsername* | username for authorization account |
| *authorizationPassword* | password for authorization account |
| *username* | username for account to set new password |
| *password* | password for account |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetConnected()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetConnected (
            VsxSystemHandle ∗ vsx,
            int32_t ∗ result )
```

Indicates current connection state with the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *result* | Pointer to result value (1: connected, 0: disconneted) |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_Disconnect()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Disconnect (
            VsxSystemHandle * vsx )
```

Disconnect with the device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_RegisterOnDisconnect()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDisconnect (
            VsxSystemHandle * vsx,
            vsx_OnDisconnect fptr )
```

Register callback for "vsx_OnDisconnect" callback. Only short execution times are allowed (do not block the function), just use to transfer data to your (main) thread.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to vsx sensor |
| *fptr* | Function pointer to callback function |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_DeregisterOnDisconnect()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDisconnect (
            VsxSystemHandle * vsx )
```

Function to deregister already existing callback function.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_RegisterOnSessionMessageReceived()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnSessionMessageReceived (
            VsxSystemHandle * vsx,
            vsx_OnSessionMessageReceived fptr )
```

Register callback for "vsx_OnSessionMessageReceived" callback. Only short execution times are allowed (do not block the function), just use to transfer data to your (main) thread.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to vsx sensor |
| *fptr* | Function pointer to callback function |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_DeregisterOnSessionMessageReceived()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnSessionMessageReceived (
            VsxSystemHandle * vsx )
```

Function to deregister already existing callback function.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SendSessionKeepAlive()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendSessionKeepAlive (
            VsxSystemHandle * vsx )
```

Send session keep alive to sensor. Should be the reply from a timeout announcement message.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_TestSystem()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystem (
             VsxSystemHandle * vsx,
             const char * command,
             const char * inputValue,
             const char ** outputValue,
             int32_t * status )
```

Sends a test system command to the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *command* | The test system command. |
| *inputValue* | Optional input value |
| *outputValue* | Return output string of function call |
| *status* | Returns 1 on success and 0 on failure |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_TestSystemEx()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystemEx (
             VsxSystemHandle * vsx,
             const char * command,
             const char * inputValue,
             const char ** outputValue,
             int32_t * status,
             int32_t timeout_ms )
```

Sends a test system command to the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *command* | The test system command. |
| *inputValue* | Optional input value |
| *outputValue* | Return ouput string of function call |
| *status* | Returns 1 on sucess and 0 on failure |
| *timeout_ms* | Wait time for device reply. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetWaitTimeout()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetWaitTimeout (
VsxSystemHandle * vsx,
int32_t * timeout_ms )
```

Gets the time in ms, the driver waits for response from device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *timeout_ms* | Time in ms |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetWaitTimeout()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetWaitTimeout (
             VsxSystemHandle * vsx,
             int32_t timeout_ms )
```

Sets the time in ms, the driver waits for response from device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *timeout_ms* | Time in ms |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_UploadData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadData (
             VsxSystemHandle * vsx,
             const char * fileName )
```

Sends a data file (either image data or dynamic container data) to the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *fileName* | The path and filename of the data file. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SendFirmware()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendFirmware (
             VsxSystemHandle * vsx,
             const char * fileName )
```

Sends a firmware update file to the device. NOTE: not completely implemented yet, the file is send only to the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *fileName* | The path and filename of the firmware file. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SendXmlDataMessage()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendXmlDataMessage (
             VsxSystemHandle * vsx,
             const char * xmlCommand )
```

Sends a string to the device. NOTE: function does not wait for any device reply.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *xmlCommand* | Command to send |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetNetworkSettings()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettings (
             VsxSystemHandle * vsx,
             const char * ipAddress,
             const char * networkMask,
             const char * gateway )
```

Sets the network settings of the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *ipAddress* | The new IP Address |
| *networkMask* | The new network mask |
| *gateway* | The new gateway |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetNetworkSettingsViaUdp()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettingsViaUdp (
             const char * macAddress,
             const char * ipAddress,
             const char * networkMask,
             const char * gateway )
```

Sets the network settings of the device identified by the macAddress via UDP.

**Parameters**

| | |
|---|---|
| *macAddress* | The macAddress of the device to set |
| *ipAddress* | The new IP Address |
| *networkMask* | The new network mask |
| *gateway* | The new gateway |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ResetDynamicContainerGrabber()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetDynamicContainerGrabber (
             VsxSystemHandle * vsx,
             int32_t bufferSize,
             VsxStrategy strategy )
```

Restarts the internal dynamic container grabber. Saving the items will be new initialized.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *bufferSize* | The maximum number of items which will be internally saved, if less than 0, number is infinity. |
| *strategy* | The strategy, which items will be discarded if maximum number of items is reached. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetDataContainer()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDataContainer (
             VsxSystemHandle * vsx,
             VsxDataContainerHandle ** pDch,
             int32_t timeout_ms )
```

Gets the oldest saved item and removes it internally.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *pDch* | New dynamic container handle |
| *timeout_ms* | The maximum time in ms to try reading an item. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetCachedContainer()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCachedContainer (
            VsxSystemHandle * vsx,
            VsxDataContainerHandle ** pDch,
            int32_t position )
```

Gets a cached dynamic container.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *pDch* | Handle to new generated data container |
| *position* | Position of the container in cache. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseDataContainer()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDataContainer (
            VsxDataContainerHandle ** dch )
```

Release / Free data container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to data container |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SaveData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveData (
            VsxDataContainerHandle * dch,
            const char * tag,
            const char * fileName )
```

Saves a VsxMessage to the given filename.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Specify which tag from container should be saved ("*" save complete container) |
| *fileName* | Path and filename where to save the message. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_Save3DPointCloudData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Save3DPointCloudData (
            VsxDataContainerHandle * dch,
            const char * point_x_Id,
            const char * point_y_Id,
            const char * point_z_Id,
            const char * fileName )
```

Saves a 3D point cloud as pcd to the given filename.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *point_x_Id* | The x image tag name |
| *point_y_Id* | The y image tag name |
| *point_z_Id* | The z image tag name |
| *fileName* | Path and filename where to save the data. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetImage()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetImage (
            VsxDataContainerHandle * dch,
            const char * tag,
            VsxImage ** imageData )
```

Get image from a dynamic container, access via raw memory pointer.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of image data |
| *imageData* | New handle to image object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseImage()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseImage (
            VsxImage ** pImage )
```

Release / free image object.

**Parameters**

| | |
|---|---|
| *pImage* | Handle to image object |

**Returns**

### vsx_GetLine()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLine (
            VsxDataContainerHandle * dch,
            const char * tag,
            VsxLineData ** data )
```

Get line data from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of line data |
| *data* | New handle to image object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseLine()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseLine (
            VsxLineData ** pLineData )
```

Release / free line data object.

**Parameters**

| | |
|---|---|
| *pLineData* | Handle to line data object |

**Returns**

### vsx_GetDisparityDescriptor2()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2 (
            VsxDataContainerHandle * dch,
            const char * tag,
            VsxDisparityDescriptor2 ** data )
```

Get disparity descriptor from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of data |
| *data* | New handle to data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseDisparityDescriptor2()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDisparityDescriptor2 (
            VsxDisparityDescriptor2 ** pData )
```

Release / free DisparityDescriptor2 object.

**Parameters**

| | |
|---|---|
| *pData* | Handle to data object |

**Returns**

### vsx_GetTransformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTransformation (
            VsxDataContainerHandle * dch,
            const char * tag,
            VsxTransformation ** data )
```

Get transformation from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of data |
| *data* | New handle to data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseTransformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTransformation (
            VsxTransformation ** pData )
```

Release / free Transformation object.

**Parameters**

| | |
|---|---|
| *pData* | Handle to data object |

**Returns**

### vsx_GetCaptureInformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCaptureInformation (
            VsxDataContainerHandle * dch,
            const char * tag,
            VsxCaptureInformation ** data )
```

Get capture information from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of data |
| *data* | New handle to data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseCaptureInformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseCaptureInformation (
            VsxCaptureInformation ** pData )
```

Release / free CaptureInformation object.

**Parameters**

| | |
|---|---|
| *pData* | Handle to data object |

**Returns**

### vsx_GetOlr2CaptureInformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2CaptureInformation (
            VsxDataContainerHandle * dch,
            const char * tag,
            VsxOlr2CaptureInformation ** data )
```

Get olr2 capture information from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of data |
| *data* | New handle to data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseOlr2CaptureInformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2CaptureInformation (
            VsxOlr2CaptureInformation ** pData )
```

Release / free Olr2CaptureInformation object.

**Parameters**

| | |
|---|---|
| *pData* | Handle to data object |

**Returns**

### vsx_GetOlr2ModbusData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2ModbusData (
             VsxDataContainerHandle * dch,
             const char * tag,
             VsxOlr2ModbusData ** data )
```

Get modbus data for olr2 sensor from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tag* | Tag name of data |
| *data* | New handle to data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseOlr2ModbusData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2ModbusData (
             VsxOlr2ModbusData ** pData )
```

Release / free Olr2ModbusData object.

**Parameters**

| | |
|---|---|
| *pData* | Handle to data object |

**Returns**

### vsx_GetTagList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTagList (
            VsxDataContainerHandle * dch,
            VsxTagList ** tagList )
```

Returns all available tags from a dynamic container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to dynamic container |
| *tagList* | New handle to tag list object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseTagList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTagList (
            VsxTagList ** pTagList )
```

Relase / free release tag list object.

**Parameters**

| | |
|---|---|
| *pTagList* | handle to tag list object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetMissingContainerFramesCounter()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingContainerFramesCounter (
            VsxSystemHandle * vsx,
            int32_t * result )
```

Gets the missing frame counter from image grabbing.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *result* | Pointer to result value |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetDynamicContainerQueueSize()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDynamicContainerQueueSize (
            VsxSystemHandle * vsx,
            int32_t * result )
```

Gets the current size of the dynamic container message queue.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *result* | Pointer to result value |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetNumberOfCachedContainers()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetNumberOfCachedContainers (
            VsxSystemHandle * vsx,
            int32_t * result )
```

Gets the current number of cached container messages.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *result* | Pointer to result value |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetDeviceInformation()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDeviceInformation (
            VsxSystemHandle * vsx,
            VsxDevice ** deviceData )
```

Returns a device object with network information about the current device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *deviceData* | New handle to device data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDevice (
            VsxDevice ** pDevice )
```

Release / Free existing handle to device object.

**Parameters**

| | |
|---|---|
| *pDevice* | Handle to existing device obeject |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetUdpDeviceList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetUdpDeviceList (
            VsxDeviceList ** deviceListData )
```

Searches for all devices in a subnet via udp and returns a list with all devices found.

**Parameters**

| | |
|---|---|
| *deviceListData* | New handle to device list object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseDeviceList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDeviceList (
            VsxDeviceList ** pDeviceList )
```

Release / Free existing handle to device list object.

**Parameters**

| | |
|---|---|
| *pDeviceList* | Handle to existing device list object |

**Returns**

### vsx_ResetLogMessageGrabber()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetLogMessageGrabber (
             VsxSystemHandle * vsx,
             int32_t bufferSize,
             int32_t typeMask,
             VsxStrategy strategy )
```

Starts the internal log message grabber. Saving the items will be new initialized.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *bufferSize* | The maximum number of items which will be internally saved, if less than 0, number is infinity. |
| *typeMask* | Mask which log message types will be send by device. |
| *strategy* | The strategy, which items will be discarded if maximum number of items is reached. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetLogMessage()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessage (
             VsxSystemHandle * vsx,
             const char ** log,
             int32_t timeout_ms )
```

Gets the oldest saved item and removes it internally.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *log* | New handle to string list |
| *timeout_ms* | The maximum time in ms to try reading an item. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetLogMessageQueueSize()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessageQueueSize (
             VsxSystemHandle * vsx,
             int32_t * result )
```

Gets the current size of the log message queue.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *result* | Pointer to result value |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetMissingLogMessagesCounter()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingLogMessagesCounter (
            VsxSystemHandle * vsx,
            int32_t * result )
```

Gets the missing log messages counter for log message grabbing.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *result* | Pointer to result value |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetSingleParameterValue()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValue (
            VsxSystemHandle * vsx,
            uint32_t settingsVersion,
            const char * configurationId,
            uint32_t configurationVersion,
            const char * parameterId,
            const char * value )
```

Sets the parameter to a value on the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *settingsVersion* | The settings version of the parameter which should be set. |
| *configurationId* | The config id of the parameter which should be set. |
| *configurationVersion* | The config version of the parameter which should be set. |
| *parameterId* | The id of the parameter which should be set. |
| *value* | Value as string (floating number must be formatted with dot separation) |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetSingleParameterValueDouble()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueDouble (
             VsxSystemHandle ∗ vsx,
             uint32_t settingsVersion,
             const char ∗ configurationId,
             uint32_t configurationVersion,
             const char ∗ parameterId,
             double value )
```

Sets the parameter to a value on the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *settingsVersion* | The settings version of the parameter which should be set. |
| *configurationId* | The config id of the parameter which should be set. |
| *configurationVersion* | The config version of the parameter which should be set. |
| *parameterId* | The id of the parameter which should be set. |
| *value* | Value as double |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetSingleParameterValueInt32()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueInt32 (
             VsxSystemHandle ∗ vsx,
             uint32_t settingsVersion,
             const char ∗ configurationId,
             uint32_t configurationVersion,
             const char ∗ parameterId,
             int32_t value )
```

Sets the parameter to a value on the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *settingsVersion* | The settings version of the parameter which should be set. |
| *configurationId* | The config id of the parameter which should be set. |
| *configurationVersion* | The config version of the parameter which should be set. |
| *parameterId* | The id of the parameter which should be set. |
| *value* | Value as int32 |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetSingleParameterValue()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValue (
             VsxSystemHandle * vsx,
             uint32_t settingsVersion,
             const char * configurationId,
             uint32_t configurationVersion,
             const char * parameterId,
             const char ** value )
```

Returns the current value of the given parameter from device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *settingsVersion* | The settings version of the parameter its value is asked for. |
| *configurationId* | The config id of the parameter its value is asked for. |
| *configurationVersion* | The config version of the parameter its value is asked for. |
| *parameterId* | The id of the parameter its value is asked for. |
| *value* | Returns value in string representation |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetSingleParameterValueDouble()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueDouble (
             VsxSystemHandle * vsx,
             uint32_t settingsVersion,
             const char * configurationId,
             uint32_t configurationVersion,
             const char * parameterId,
             double * value )
```

Returns the current value of the given parameter from device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *settingsVersion* | The settings version of the parameter its value is asked for. |
| *configurationId* | The config id of the parameter its value is asked for. |
| *configurationVersion* | The config version of the parameter its value is asked for. |
| *parameterId* | The id of the parameter its value is asked for. |
| *value* | Returns value in double reprasentation |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetSingleParameterValueInt32()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueInt32 (
            VsxSystemHandle * vsx,
            uint32_t settingsVersion,
            const char * configurationId,
            uint32_t configurationVersion,
            const char * parameterId,
            int32_t * value )
```

Returns the current value of the given parameter from device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *settingsVersion* | The settings version of the parameter its value is asked for. |
| *configurationId* | The config id of the parameter its value is asked for. |
| *configurationVersion* | The config version of the parameter its value is asked for. |
| *parameterId* | The id of the parameter its value is asked for. |
| *value* | Returns value in int32 reprasentation |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_LoadDefaultParameterSetOnDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadDefaultParameterSetOnDevice (
            VsxSystemHandle * vsx )
```

Resets the devices parameters to factory settings and returns a list of the complete parameter set of the device including current values.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_LoadParameterSetOnDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadParameterSetOnDevice (
            VsxSystemHandle * vsx )
```

Loads the parameter set saved on device and returns a list of the complete parameter set of the device including current values.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SaveParameterSetOnDevice()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveParameterSetOnDevice (
            VsxSystemHandle * vsx )
```

Saves the current parameter set on device. Parameter values will be loaded when device starts.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_UploadParameterSet()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterSet (
            VsxSystemHandle * vsx,
            const char * fileName )
```

Uploads a parameter file to the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *fileName* | Path and filename to upload. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_DownloadParameterSet()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DownloadParameterSet (
            VsxSystemHandle * vsx,
            const char * fileName )
```

Save the current parameter set to a file.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *fileName* | Path and file name to save to. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetParameterList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetParameterList (
            VsxSystemHandle * vsx,
            VsxParameterList ** parameterListData )
```

Returns a list of the complete parameter set of the device including their current values. The list shows the current state of the parameters.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *parameterListData* | New handle to parameter list data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_UploadParameterList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterList (
            VsxSystemHandle * vsx,
            VsxParameterList * parameterListData )
```

Uploads a parameter list to the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *parameterListData* | Existing handle to parameter list data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetSingleParameterString()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterString (
            VsxSystemHandle * vsx,
            const VsxParameter * parameter,
            const char * value )
```

Sets the parameter to a value on the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *parameter* | The parameter the value should be set from. |
| *value* | New value to set. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetSingleParameterDouble()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble (
             VsxSystemHandle * vsx,
             const VsxParameter * parameter,
             double value )
```

Sets the parameter to a value on the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *parameter* | The parameter the value should be set from. |
| *value* | New value to set. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SetSingleParameterInt32()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterInt32 (
             VsxSystemHandle * vsx,
             const VsxParameter * parameter,
             int32_t value )
```

Sets the parameter to a value on the device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *parameter* | The parameter the value should be set from. |
| *value* | New value to set. |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetSingleParameter()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameter (
             VsxSystemHandle * vsx,
             const VsxParameter * parameterIn,
             const VsxParameter ** parameterOut )
```

Returns the current value of the given parameter from device.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |
| *parameterIn* | The parameter its value is asked for |
| *parameterOut* | The new parameter, which must be freed with 'vsx_ReleaseParameter' function |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseParameter()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameter (
             const VsxParameter ** pParameter )
```

Release parameter handle.

**Parameters**

| | |
|---|---|
| *pParameter* | Handle to parameter |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseParameterList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameterList (
             VsxParameterList ** pParameterList )
```

Release parameter list handle.

**Parameters**

| | |
|---|---|
| *pParameterList* | Handle to parameter list |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetResultXml()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultXml (
             VsxDataContainerHandle * dch,
             const char * resultId,
             const char ** result )
```

Returns the complete xml response from an result inside data container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to data container |
| *resultId* | Name of result |
| *result* | Returns complete result xml as string |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetResultElementString()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementString (
             VsxDataContainerHandle * dch,
             const char * resultId,
             const char * xPath,
             const char ** result )
```

Return certain value from a result inside data container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to data container |
| *resultId* | Name of result |
| *xPath* | xPath definition |
| *result* | Return result as string |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetResultElementInt32()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32 (
             VsxDataContainerHandle * dch,
             const char * resultId,
             const char * xPath,
             int32_t * result )
```

Return certain value from a result inside data container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to data container |
| *resultId* | Name of result |
| *xPath* | xPath defintion |
| *result* | Return result as int32 |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetResultElementInt64()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt64 (
            VsxDataContainerHandle * dch,
            const char * resultId,
            const char * xPath,
            LOCAL_INT64_T * result )
```

Return certain value from a result inside data container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to data container |
| *resultId* | Name of result |
| *xPath* | xPath defintion |
| *result* | Return result as int64 |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetResultElementDouble()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementDouble (
            VsxDataContainerHandle * dch,
            const char * resultId,
            const char * xPath,
            double * result )
```

Return certain value from a result inside data container.

**Parameters**

| | |
|---|---|
| *dch* | Handle to data container |
| *result↩ Id* | Name of result |
| *xPath* | xPath defintion |
| *result* | Return result as double |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_GetAllDeviceStatusData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetAllDeviceStatusData (
            VsxSystemHandle * vsx,
            VsxStatusItemList ** statusItemListData )
```

Get the full status data set from device.

**Parameters**

| | |
|---|---|
| *vsx* | handle to sensor |
| *statusItemListData* | new handle to status data object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_ReleaseStatusItemList()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseStatusItemList (
            VsxStatusItemList ** pStatusItemList )
```

Release / Free "VsxStatusItemList" handle.

**Parameters**

| | |
|---|---|
| *pStatusItemList* | handle to status item object |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_RegisterOnDeviceStatusReceived()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDeviceStatusReceived (
            VsxSystemHandle * vsx,
            vsx_OnDeviceStatusReceived fptr )
```

Register callback for "vsx_OnDeviceStatusReceived" callback. Only short execution times are allowed (do not block the function), just use to transfer data to your (main) thread.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to vsx sensor |
| *fptr* | Function pointer to callback function |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_DeregisterOnDeviceStatusReceived()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDeviceStatusReceived (
             VsxSystemHandle * vsx )
```

Function to deregister already existing callback function.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_SubscribeToDeviceStatusData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SubscribeToDeviceStatusData (
             VsxSystemHandle * vsx )
```

Subscribe status data from sensor to the client. This will send periodically or in case of a problem status data to the client. This need a registered callback for "vsx_OnDeviceStatusReceived".

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

### vsx_UnsubscribeToDeviceStatusData()

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UnsubscribeToDeviceStatusData (
             VsxSystemHandle * vsx )
```

Unsubscribe status data from sensor.

**Parameters**

| | |
|---|---|
| *vsx* | Handle to sensor |

**Returns**

Returns VSX_STATUS_SUCCESS(0) on success

## 9.9 PF.VsxProtocolDriver.WrapperNE.h

Go to the documentation of this file.

```
//
// Auto-generated by dnne-gen
//
// .NET Assembly: PF.VsxProtocolDriver.Wrapper
//

//
// Declare exported functions
//
#ifndef __DNNE_GENERATED_HEADER_PF_VSXPROTOCOLDRIVER_WRAPPER__
#define __DNNE_GENERATED_HEADER_PF_VSXPROTOCOLDRIVER_WRAPPER__

#include <stddef.h>
#include <stdint.h>
#include <dnne.h>

//
// Additional code provided by user
//

#ifdef _CVI_
typedef struct _big64
        {
            unsigned int x;
            unsigned int y;
        }
        big64;
#define LOCAL_INT64_T big64
typedef struct _unsigned_big64
        {
            unsigned int x;
            unsigned int y;
        }
        unsigned_big64;
#define LOCAL_UINT64_T unsigned_big64
#else
#define LOCAL_INT64_T int64_t
#define LOCAL_UINT64_T uint64_t
#endif

typedef enum _vsxStrategy
        {
            VSX_STRATEGY_DROP_OLDEST = 0,
            VSX_STRATEGY_DROP_WRITE = 1
        }
        VsxStrategy;

typedef enum _vsxStatusCode {
        VSX_STATUS_SUCCESS = 0,

        VSX_STATUS_ERROR_DRIVER_INIT = -0x1,
        VSX_STATUS_ERROR_DRIVER_TIMEOUT = -0x2,
        VSX_STATUS_ERROR_DRIVER_SAVE_FILE = -0x3,
        VSX_STATUS_ERROR_DRIVER_DATA = -0x4,
        VSX_STATUS_ERROR_DRIVER_CONNECTION = -0x5,
        VSX_STATUS_ERROR_DRIVER_INVALID_DATA = -0x6,
        VSX_STATUS_ERROR_DRIVER_DEVICE = -0x7,
        VSX_STATUS_ERROR_DRIVER_LOAD_FILE = -0x8,
        VSX_STATUS_ERROR_SESSION = -0x9,
        VSX_STATUS_ERROR_STRING = -0x0A,
        VSX_STATUS_ERROR_VERSION = -0x0B,
        VSX_STATUS_ERROR_DRIVER_GENERAL = -0x1000,

        VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_SYSTEM = -0x8001,
        VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_ZERO = -0x8002,
        VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_ZERO = -0x8003,
        VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_AVAILABLE = -0x8004,
        VSX_STATUS_ERROR_MISSING_IP_ADDRESS_DECLARATION = -0x8005,
        VSX_STATUS_ERROR_MISSING_SERIALPORT_DECLARATION = -0x8006,
        VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_POINTER_ZERO = -0x8007,
        VSX_STATUS_ERROR_CONFIGURATION_ID_ZERO = -0x8008,
        VSX_STATUS_ERROR_PARAMETER_ID_ZERO = -0x8009,
        VSX_STATUS_ERROR_VALUE_ZERO = -0x800A,
        VSX_STATUS_ERROR_COMMAND_ZERO = -0x800B,
        VSX_STATUS_ERROR_INPUT_VALUE_ZERO = -0x800C,
        VSX_STATUS_ERROR_OUTPUT_VALUE_POINTER_ZERO = -0x800D,
        VSX_STATUS_ERROR_OUTPUT_VALUE_NOT_ZERO = -0x800E,
        VSX_STATUS_ERROR_VALUE_POINTER_ZERO = -0x800F,
        VSX_STATUS_ERROR_VALUE_NOT_ZERO = -0x8010,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX_SYSTEM = -0x8011,
        VSX_STATUS_ERROR_XML_COMMAND_ZERO = -0x8012,
        VSX_STATUS_ERROR_FILENAME_ZERO = -0x8013,
        VSX_STATUS_ERROR_STRING_POINTER_ZERO = -0x8014,
        VSX_STATUS_ERROR_STRING_ZERO = -0x8015,
        VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_POINTER_ZERO = -0x8016,
        VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_DATA_CONTAINER = -0x8017,
        VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_NOT_ZERO = -0x8018,
        VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_ZERO = -0x8019,
        VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_NOT_AVAILABLE = -0x801A,
        VSX_STATUS_ERROR_IMAGE_TAG_ZERO = -0x801B,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX_DATA_CONTAINER = -0x801C,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_IMAGE_ID_IN_DATA_CONTAINER = -0x801D,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_IMAGE_TAG_TO_DATA_FORMAT = -0x801E,
        VSX_STATUS_ERROR_POINT_Z_ID_ZERO = -0x801F,
        VSX_STATUS_ERROR_POINT_Y_ID_ZERO = -0x8020,
        VSX_STATUS_ERROR_POINT_X_ID_ZERO = -0x8021,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_X_ID_IN_DATA_CONTAINER = -0x8022,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Y_ID_IN_DATA_CONTAINER = -0x8023,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_IN_DATA_CONTAINER = -0x8024,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_X_ID_TO_DATA_FORMAT = -0x8025,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Y_ID_TO_DATA_FORMAT = -0x8026,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_TO_DATA_FORMAT = -0x8027,
        VSX_STATUS_ERROR_LOG_POINTER_ZERO = -0x8028,
        VSX_STATUS_ERROR_LOG_NOT_ZERO = -0x8029,
        VSX_STATUS_ERROR_RESULT_NOT_ZERO = -0x802A,
        VSX_STATUS_ERROR_RESULT_POINTER_ZERO = -0x802B,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_RESULT_ID_IN_DATA_CONTAINER = -0x802C,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_RESULT_ID_TO_DATA_FORMAT = -0x802D,
        VSX_STATUS_ERROR_VERSION_POINTER_ZERO = -0x802E,
        VSX_STATUS_ERROR_VERSION_NOT_ZERO = -0x802F,
        VSX_STATUS_ERROR_VSX_IMAGE_POINTER_ZERO = -0x8030,
        VSX_STATUS_ERROR_VSX_IMAGE_NOT_ZERO = -0x8031,
        VSX_STATUS_ERROR_UNDEFINED_STRATEGY_VALUE = -0x8032,
        VSX_STATUS_ERROR_UNDEFINED_CONNECTION_TYPE_VALUE = -0x8033,
        VSX_STATUS_ERROR_XPATH_ZERO = -0x8034,
        VSX_STATUS_ERROR_INVALID_DATA_FORMAT = -0x8035,
        VSX_STATUS_ERROR_NO_ELEMENT_FOUND = -0x8036,
        VSX_STATUS_ERROR_RESULT_TAG_ZERO = -0x8037,
        VSX_STATUS_ERROR_TAG_ZERO = -0x8038,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG_IN_DATA_CONTAINER = -0x8039,
        VSX_STATUS_ERROR_IP_ADDRESS_ZERO = -0x803A,
        VSX_STATUS_ERROR_NETWORK_MASK_ZERO = -0x803B,
        VSX_STATUS_ERROR_GATEWAY_ZERO = -0x803C,
        VSX_STATUS_ERROR_EXCEPTION_THROWN = -0x803D,
        VSX_STATUS_ERROR_VSX_DEVICE_POINTER_ZERO = -0x803E,
        VSX_STATUS_ERROR_VSX_DEVICE_NOT_ZERO = -0x803F,
        VSX_STATUS_ERROR_VSX_IMAGE_ZERO = -0x8040,
        VSX_STATUS_ERROR_VSX_DEVICE_ZERO = -0x8041,
        VSX_STATUS_ERROR_VSX_DEVICE_LIST_POINTER_ZERO = -0x8042,
        VSX_STATUS_ERROR_VSX_DEVICE_LIST_ZERO = -0x8043,
        VSX_STATUS_ERROR_VSX_TAG_LIST_ZERO = -0x8044,
        VSX_STATUS_ERROR_VSX_TAG_LIST_POINTER_ZERO = -0x8045,
        VSX_STATUS_ERROR_VSX_TAG_LIST_NOT_ZERO = -0x8046,
        VSX_STATUS_ERROR_VSX_PARAMETER_LIST_POINTER_ZERO = -0x8047,
        VSX_STATUS_ERROR_VSX_PARAMETER_LIST_ZERO = -0x8048,
        VSX_STATUS_ERROR_VSX_PARAMETER_NOT_ZERO = -0x8049,
        VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_POINTER_ZERO = -0x804A,
        VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_ZERO = -0x804B,
        VSX_STATUS_ERROR_VSX_STATUS_ITEM_NOT_ZERO = -0x0804C,
        VSX_STATUS_ERROR_ERROR_TEXT_POINTER_ZERO = -0x0804D,
        VSX_STATUS_ERROR_ERROR_TEXT_NOT_ZERO = -0x0804E,
        VSX_STATUS_ERROR_ON_DISCONNECT_CALLBACK_ZERO = -0x0804F,
        VSX_STATUS_ERROR_MAC_ADDRESS_ZERO = -0x08050,
        VSX_STATUS_ERROR_VSX_CACHED_CONTAINER_NOT_FOUND = -0x08051,
        VSX_STATUS_ERROR_VSX_PARAMETER_LIST_NOT_ZERO = -0x08052,
        VSX_STATUS_ERROR_VSX_PARAMETER_POINTER_ZERO = -0x08053,
        VSX_STATUS_ERROR_VSX_PARAMETER_ZERO = -0x08054,
        VSX_STATUS_ERROR_VSX_LINE_DATA_POINTER_ZERO = -0x08064,
        VSX_STATUS_ERROR_LINE_DATA_TAG_ZERO = -0x08065,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_ID_IN_DATA_CONTAINER = -0x08066,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_TAG_TO_DATA_FORMAT = -0x08067,
        VSX_STATUS_ERROR_VSX_LINE_NOT_ZERO = -0x08068,
        VSX_STATUS_ERROR_VSX_LINE_DATA_ZERO = -0x08069,
        VSX_STATUS_ERROR_MISSING_LOGIN_PASSWORD = -0x0806A,
        VSX_STATUS_ERROR_MISSING_LOGIN_USERNAME = -0x0806B,
        VSX_STATUS_ERROR_ON_SESSION_MESSAGE_RECEIVED_CALLBACK_ZERO = -0x0806C,
        VSX_STATUS_ERROR_VSX_PARAMETER_IN_POINTER_ZERO = -0x0806D,
        VSX_STATUS_ERROR_VSX_PARAMETER_OUT_POINTER_ZERO = -0x0806E,
        VSX_STATUS_ERROR_VSX_VALUE_POINTER_ZERO = -0x0806F,
        VSX_STATUS_ERROR_DATA_POINTER_ZERO = -0x08070,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_MESSAGE_IN_DATA_CONTAINER = -0x08071,
        VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_NOT_ZERO = -0x08072,
        VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_ZERO = -0x08073,
        VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG = -0x08074,
        VSX_STATUS_ERROR_INCORRECT_MESSAGE_FROM_TAG = -0x08075,
} VsxStatusCode;

typedef enum _vsxSerialConnectionType {
    VSX_SERIAL_CONNECTION_TYPE_USB_SSI = 0,
    VSX_SERIAL_CONNECTION_TYPE_PROFIBUS = 1,
    VSX_SERIAL_CONNECTION_TYPE_PROFINET = 2,
    VSX_SERIAL_CONNECTION_TYPE_ETHERNET_IP = 3,
    VSX_SERIAL_CONNECTION_TYPE_RS485 = 4,
    VSX_SERIAL_CONNECTION_TYPE_CANOPEN = 5
} VsxSerialConnectionType;


        typedef struct _VsxSystemHandle
        {
            int handle;
        } VsxSystemHandle;


typedef enum _vsxDisconnectEvent {
    VSX_DISCONNECT_EVENT_REMOTE_HOST_CONNECTION_CLOSED = 0,
    VSX_DISCONNECT_EVENT_DISCONNECT_CALLED = 1,
    VSX_DISCONNECT_EVENT_CONNECTION_ERROR = 2
} VsxDisconnectEvent;

typedef void (*vsx_OnDisconnect) (int handle, const char* ipAddress, VsxDisconnectEvent disconnectEvent, const char* description);


typedef enum _vsxSessionTypes {
    VSX_SESSION_TYPES_LOGIN_REQUIRED = 0,
    VSX_SESSION_TYPES_INITIAL_PASSWORD_REQUIRED = 1,
    VSX_SESSION_TYPES_LOGIN = 2,
    VSX_SESSION_TYPES_LOGIN_REPLY = 3,
    VSX_SESSION_TYPES_SET_PASSWORD = 4,
    VSX_SESSION_TYPES_SET_PASSWORD_REPLY = 5,
    VSX_SESSION_TYPES_TIMEOUT_ANNOUNCEMENT = 6,
    VSX_SESSION_TYPES_TIMEOUT = 7,
    VSX_SESSION_TYPES_LOGOUT = 8,
    VSX_SESSION_TYPES_LOGOUT_REPLY = 9,
    VSX_SESSION_TYPES_UNKNOWN = 10
} VsxSessionTypes;

typedef void (*vsx_OnSessionMessageReceived) (int handle, VsxSessionTypes sessionType, int timeout);


typedef struct _VsxDataContainerHandle
{
    int handle;
} VsxDataContainerHandle;

typedef enum _vsxImageData2Format {
    VSX_IMAGE_DATA2_FORMAT_MONO8 = 17301505,
    VSX_IMAGE_DATA2_FORMAT_CONFIDENCE8 = 17301702,
    VSX_IMAGE_DATA2_FORMAT_MONO12 = 17825797,
    VSX_IMAGE_DATA2_FORMAT_MONO16 = 17825799,
    VSX_IMAGE_DATA2_FORMAT_COORD3D_A16 = 17825974,
    VSX_IMAGE_DATA2_FORMAT_COORD3D_B16 = 17825975,
    VSX_IMAGE_DATA2_FORMAT_COORD3D_C16 = 17825976,
    VSX_IMAGE_DATA2_FORMAT_COORD3D_A32F = 18874557,
    VSX_IMAGE_DATA2_FORMAT_COORD3D_B32F = 18874558,
    VSX_IMAGE_DATA2_FORMAT_COORD3D_C32F = 18874559
} VsxImageData2Format;

typedef struct _VsxImage{
    void* rawdata;
    VsxImageData2Format format;
    int width;
    int height;
    int linePitch;
    LOCAL_INT64_T frameCounter;
    double coordinateScale;
    double coordinateOffset;
    double axisMin;
    double axisMax;
    double invalidDataValue;
} VsxImage;
```

```


typedef enum _vsxLineDataFormat {
    VSX_LINE_DATA_FORMAT_C = 0x00,
    VSX_LINE_DATA_FORMAT_X = 0x01,
    VSX_LINE_DATA_FORMAT_Y = 0x02,
    VSX_LINE_DATA_FORMAT_Z = 0x04,
    VSX_LINE_DATA_FORMAT_Q = 0x08,
    VSX_LINE_DATA_FORMAT_I = 0x10,
    VSX_LINE_DATA_FORMAT_32BIT_MODE = 0x1000,
} VsxLineDataFormat;

typedef struct _VsxLineCoordinate{
    float c;
    float x;
    float y;
    float z;
    float q;
    float i;
} VsxLineCoordinate;

typedef struct _VsxLineData{
    VsxLineCoordinate** lines;
    unsigned short format;
    unsigned short width;
    unsigned short countLines;
    unsigned short frameCounter;
    float minX;
    float maxX;
    float minZ;
    float maxZ;
} VsxLineData;


typedef struct _VsxDisparityDescriptor2{
    double focalLength;
    double principalPointU;
    double principalPointV;
    double baseline;
    double offsetLeftRectifiedToDisparityU;
    double offsetLeftRectifiedToDisparityV;
} VsxDisparityDescriptor2;


typedef struct _VsxTransformation{
    double translationTX;
    double translationTY;
    double translationTZ;
    double quaternionQ0;
    double quaternionQ1;
    double quaternionQ2;
    double quaternionQ3;
} VsxTransformation;


typedef struct _VsxCaptureInformation{
    LOCAL_UINT64_T triggerCounter;
    LOCAL_UINT64_T parameterId;
    LOCAL_UINT64_T jobId;
    LOCAL_INT64_T rotaryEncoder;
    LOCAL_UINT64_T frameCounter;
    LOCAL_UINT64_T timestamp;
    unsigned int exposureTime;
    unsigned int gain;
    unsigned char illumination;
    unsigned char triggerSource;
} VsxCaptureInformation;


typedef struct _VsxOlr2CaptureInformation {
    LOCAL_UINT64_T frameCounter;
    LOCAL_UINT64_T triggerCounter;
    double currentPosition;
    LOCAL_UINT64_T ioState;
    LOCAL_UINT64_T timestamp;
    unsigned int lmaExposureTime1;
    unsigned int lmaExposureTime2;
    unsigned int lmbExposureTime1;
    unsigned int lmbExposureTime2;
    unsigned short lmaRoiOffsetX;
    unsigned short lmaRoiLengthX;
    unsigned short lmaRoiOffsetZ;
    unsigned short lmaRoiLengthZ;
    unsigned short lmbRoiOffsetX;
    unsigned short lmbRoiLengthX;
    unsigned short lmbRoiOffsetZ;
    unsigned short lmbRoiLengthZ;
    unsigned short autoTriggerFrameRate ;
    unsigned char triggerSource;
} VsxOlr2CaptureInformation;


typedef struct _VsxOlr2ModbusData{
    unsigned short activationTimer;
    unsigned short compareBuffer;
    unsigned short targetPosition;
    unsigned short robotData[13];
} VsxOlr2ModbusData;


typedef struct _VsxTagList{
    int length;
    const char** tags;
} VsxTagList;


typedef struct _VsxDevice{
    const char* ipAddress;
    const char* networkMask;
    const char* gateway;
    const char* macAddress;
    const char* firmwareVersion;
    const char* sensorType;
    const char* sensorName;
    int busy;
    int deviceVsxVersionMajor;
    int deviceVsxVersionMinor;
    const char* comPort;
    int baudrate;
    const char* headAddress;
    int isLoginNeeded;
} VsxDevice;

typedef struct _VsxDeviceList{
    int length;
    const VsxDevice* devices;
} VsxDeviceList;


typedef enum _vsxParameterValueType {
    VSX_PARAMETER_VALUE_TYPE_BOOL = 0,
    VSX_PARAMETER_VALUE_TYPE_INT = 1,
    VSX_PARAMETER_VALUE_TYPE_LONG = 2,
    VSX_PARAMETER_VALUE_TYPE_UINT = 3,
    VSX_PARAMETER_VALUE_TYPE_INT16 = 4,
    VSX_PARAMETER_VALUE_TYPE_FLOAT = 5,
    VSX_PARAMETER_VALUE_TYPE_DOUBLE = 6,
    VSX_PARAMETER_VALUE_TYPE_STRING = 7,
    VSX_PARAMETER_VALUE_TYPE_HEXSTRING = 8,
    VSX_PARAMETER_VALUE_TYPE_BASE64 = 9,
    VSX_PARAMETER_VALUE_TYPE_ENUM = 10,
    VSX_PARAMETER_VALUE_TYPE_IP = 11,
    VSX_PARAMETER_VALUE_TYPE_RECTANGLE = 12,
    VSX_PARAMETER_VALUE_TYPE_QUAD = 13,
    VSX_PARAMETER_VALUE_TYPE_POINT = 14,
    VSX_PARAMETER_VALUE_TYPE_UNKNOWN = 15
} VsxParameterValueType;

typedef struct _VsxParameterEnumItem{
    const char* id;
    const char* name;
} VsxParameterEnumItem;

typedef struct _VsxParameter{
    const char* valueString;
    int valueInt;
    double valueDouble;
    VsxParameterValueType valueType;
    const char* name;
    const char* parameterId;
    const char* configId;
    int configVersion;
    int settingsVersion;
    int enumItemListLength;
    const VsxParameterEnumItem* enumItemList;
} VsxParameter;

typedef struct _VsxParameterList{
    int length;
    const VsxParameter* parameters;
} VsxParameterList;


typedef enum _vsxStatusItemValueType {
    VSX_STATUS_ITEM_VALUE_TYPE_BOOL = 0,
    VSX_STATUS_ITEM_VALUE_TYPE_INT = 1,
    VSX_STATUS_ITEM_VALUE_TYPE_LONG = 2,
    VSX_STATUS_ITEM_VALUE_TYPE_UINT = 3,
    VSX_STATUS_ITEM_VALUE_TYPE_INT16 = 4,
    VSX_STATUS_ITEM_VALUE_TYPE_FLOAT = 5,
    VSX_STATUS_ITEM_VALUE_TYPE_DOUBLE = 6,
    VSX_STATUS_ITEM_VALUE_TYPE_STRING = 7,
    VSX_STATUS_ITEM_VALUE_TYPE_HEXSTRING = 8,
    VSX_STATUS_ITEM_VALUE_TYPE_BASE64 = 9,
    VSX_STATUS_ITEM_VALUE_TYPE_ENUM = 10,
    VSX_STATUS_ITEM_VALUE_TYPE_IP = 11,
    VSX_STATUS_ITEM_VALUE_TYPE_RECTANGLE = 12,
    VSX_STATUS_ITEM_VALUE_TYPE_QUAD = 13,
    VSX_STATUS_ITEM_VALUE_TYPE_POINT = 14,
    VSX_STATUS_ITEM_VALUE_TYPE_UNKNOWN = 15
} VsxStatusItemValueType;

typedef enum _vsxDeviceStatusScope {
    VSX_DEVICE_STATUS_SCOPE_FULL = 0,
    VSX_DEVICE_STATUS_SCOPE_MULTI = 1
} VsxDeviceStatusScope;

typedef struct _VsxStatusItem{
    const char* valueString;
    int valueInt;
    double valueDouble;
    VsxStatusItemValueType valueType;
    const char* name;
    const char* statusItemId;
    const char* configurationClass;
    int configVersion;
    int settingsVersion;
    LOCAL_UINT64_T time;
    LOCAL_UINT64_T sensorTime;
} VsxStatusItem;

typedef struct _VsxStatusItemList{
    int length;
    const VsxStatusItem* statusItems;
} VsxStatusItemList;


typedef void (*vsx_OnDeviceStatusReceived) (int handle, VsxDeviceStatusScope deviceStatusScope, const VsxStatusItemList* statusItemListData);


// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseString
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseString(const char** pString);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetLibraryVersion
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLibraryVersion(const char** version);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetErrorText
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetErrorText(int32_t error_code, const char** error_text);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitTcpSensor
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitTcpSensor(VsxSystemHandle** pVsx, const char* ipAddress, const char* pluginName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitSerialSensor
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitSerialSensor(VsxSystemHandle** pVsx, const char* serialPort, int32_t baudrate, const char* sensorType, VsxSerialConnectionType connectionType, const char* pluginName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseSensor
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseSensor(VsxSystemHandle** vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectTcpDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectTcpDevice(VsxSystemHandle* vsx, const char* ipAddress);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectAndLoginTcpDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectAndLoginTcpDevice(VsxSystemHandle* vsx, const char* ipAddress, const char* username, const char* password);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectSerialDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectSerialDevice(VsxSystemHandle* vsx, const char* serialPort, int32_t baudrate, VsxSerialConnectionType connectionType);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Connect
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Connect(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectEx
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectEx(VsxSystemHandle* vsx, int32_t timeout_ms);
```

```

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectAndLogin
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectAndLogin(VsxSystemHandle* vsx, const char* username, const char* password);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectExAndLogin
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectExAndLogin(VsxSystemHandle* vsx, const char* username, const char* password, int32_t timeout_ms);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Login
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Login(VsxSystemHandle* vsx, const char* username, const char* password);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Logout
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Logout(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetPassword
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetPassword(VsxSystemHandle* vsx, const char* authorizationUsername, const char* authorizationPassword, const char* username, const char* password);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetConnected
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetConnected(VsxSystemHandle* vsx, int32_t* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Disconnect
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Disconnect(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnDisconnect
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDisconnect(VsxSystemHandle* vsx, vsx_OnDisconnect fptr);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnDisconnect
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDisconnect(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnSessionMessageReceived
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnSessionMessageReceived(VsxSystemHandle* vsx, vsx_OnSessionMessageReceived fptr);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnSessionMessageReceived
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnSessionMessageReceived(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendSessionKeepAlive
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendSessionKeepAlive(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystem
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystem(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystemEx
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystemEx(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status, int32_t timeout_ms);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetWaitTimeout
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetWaitTimeout(VsxSystemHandle* vsx, int32_t* timeout_ms);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetWaitTimeout
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetWaitTimeout(VsxSystemHandle* vsx, int32_t timeout_ms);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.UploadData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadData(VsxSystemHandle* vsx, const char* fileName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendFirmware
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendFirmware(VsxSystemHandle* vsx, const char* fileName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendXmlDataMessage
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendXmlDataMessage(VsxSystemHandle* vsx, const char* xmlCommand);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettings
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettings(VsxSystemHandle* vsx, const char* ipAddress, const char* networkMask, const char* gateway);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettingsViaUdp
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettingsViaUdp(const char* macAddress, const char* ipAddress, const char* networkMask, const char* gateway);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ResetDynamicContainerGrabber
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetDynamicContainerGrabber(VsxSystemHandle* vsx, int32_t bufferSize, VsxStrategy strategy);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDataContainer
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDataContainer(VsxSystemHandle* vsx,
VsxDataContainerHandle** pDch, int32_t timeout_ms);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCachedContainer
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCachedContainer(VsxSystemHandle* vsx,
VsxDataContainerHandle** pDch, int32_t position);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDataContainer
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDataContainer(VsxDataContainerHandle**
dch);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.SaveData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveData(VsxDataContainerHandle* dch, const
char* tag, const char* fileName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.Save3DPointCloudData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Save3DPointCloudData(VsxDataContainerHandle*
dch, const char* point_x_Id, const char* point_y_Id, const char* point_z_Id, const char* fileName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetImage
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetImage(VsxDataContainerHandle* dch, const
char* tag, VsxImage** imageData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseImage
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseImage(VsxImage** pImage);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetLine
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLine(VsxDataContainerHandle* dch, const
char* tag, VsxLineData** data);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseLine
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseLine(VsxLineData** pLineData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDisparityDescriptor2
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2(VsxDataContainerHandle*
dch, const char* tag, VsxDisparityDescriptor2** data);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDisparityDescriptor2
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseDisparityDescriptor2(VsxDisparityDescriptor2** pData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTransformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTransformation(VsxDataContainerHandle* dch,
const char* tag, VsxTransformation** data);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTransformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTransformation(VsxTransformation**
pData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCaptureInformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCaptureInformation(VsxDataContainerHandle*
dch, const char* tag, VsxCaptureInformation** data);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseCaptureInformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseCaptureInformation(VsxCaptureInformation** pData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2CaptureInformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_GetOlr2CaptureInformation(VsxDataContainerHandle* dch, const char* tag,
VsxOlr2CaptureInformation** data);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2CaptureInformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseOlr2CaptureInformation(VsxOlr2CaptureInformation** pData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2ModbusData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2ModbusData(VsxDataContainerHandle* dch,
const char* tag, VsxOlr2ModbusData** data);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2ModbusData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2ModbusData(VsxOlr2ModbusData**
pData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTagList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTagList(VsxDataContainerHandle* dch,
VsxTagList** tagList);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTagList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTagList(VsxTagList** pTagList);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetMissingContainerFramesCounter
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_GetMissingContainerFramesCounter(VsxSystemHandle* vsx, int32_t* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDynamicContainerQueueSize
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDynamicContainerQueueSize(VsxSystemHandle*
vsx, int32_t* result);
```

```

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetNumberOfCachedContainers
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetNumberOfCachedContainers(VsxSystemHandle* vsx, int32_t* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetDeviceInformation
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDeviceInformation(VsxSystemHandle* vsx, VsxDevice** deviceData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDevice(VsxDevice** pDevice);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetUdpDeviceList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetUdpDeviceList(VsxDeviceList** deviceListData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDeviceList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDeviceList(VsxDeviceList** pDeviceList);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.ResetLogMessageGrabber
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetLogMessageGrabber(VsxSystemHandle* vsx, int32_t bufferSize, int32_t typeMask, VsxStrategy strategy);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessage
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessage(VsxSystemHandle* vsx, const char** log, int32_t timeout_ms);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessageQueueSize
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessageQueueSize(VsxSystemHandle* vsx, int32_t* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetMissingLogMessagesCounter
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingLogMessagesCounter(VsxSystemHandle* vsx, int32_t* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValue
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValue(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, const char* value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueDouble
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueDouble(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, double value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueInt32
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueInt32(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, int32_t value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValue
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValue(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, const char** value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueDouble
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueDouble(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, double* value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueInt32
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueInt32(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, int32_t* value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadDefaultParameterSetOnDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadDefaultParameterSetOnDevice(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadParameterSetOnDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadParameterSetOnDevice(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SaveParameterSetOnDevice
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveParameterSetOnDevice(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.UploadParameterSet
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterSet(VsxSystemHandle* vsx, const char* fileName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.DownloadParameterSet
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DownloadParameterSet(VsxSystemHandle* vsx, const char* fileName);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetParameterList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetParameterList(VsxSystemHandle* vsx, VsxParameterList** parameterListData);

```

```
// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.UploadParameterList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterList(VsxSystemHandle* vsx, VsxParameterList* parameterListData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterString
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterString(VsxSystemHandle* vsx, const VsxParameter* parameter, const char* value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterDouble
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble(VsxSystemHandle* vsx, const VsxParameter* parameter, double value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterInt32
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterInt32(VsxSystemHandle* vsx, const VsxParameter* parameter, int32_t value);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetSingleParameter
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameter(VsxSystemHandle* vsx, const VsxParameter* parameterIn, const VsxParameter** parameterOut);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameter
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameter(const VsxParameter** pParameter);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameterList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameterList(VsxParameterList** pParameterList);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultXml
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultXml(VsxDataContainerHandle* dch, const char* resultId, const char** result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementString
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementString(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, const char** result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt32
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, int32_t* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt64
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt64(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, LOCAL_INT64_T* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementDouble
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementDouble(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, double* result);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.GetAllDeviceStatusData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetAllDeviceStatusData(VsxSystemHandle* vsx, VsxStatusItemList** statusItemListData);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.ReleaseStatusItemList
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseStatusItemList(VsxStatusItemList** pStatusItemList);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.RegisterOnDeviceStatusReceived
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDeviceStatusReceived(VsxSystemHandle* vsx, vsx_OnDeviceStatusReceived fptr);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.DeregisterOnDeviceStatusReceived
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDeviceStatusReceived(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.SubscribeToDeviceStatusData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SubscribeToDeviceStatusData(VsxSystemHandle* vsx);

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.UnsubscribeToDeviceStatusData
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UnsubscribeToDeviceStatusData(VsxSystemHandle* vsx);

#endif // __DNNE_GENERATED_HEADER_PF_VSXPROTOCOLDRIVER_WRAPPER__

//
// Define exported functions
//
#ifdef DNNE_COMPILE_AS_SOURCE

#ifdef DNNE_WINDOWS
    #ifdef _WCHAR_T_DEFINED
        typedef wchar_t char_t;
    #else
        typedef unsigned short char_t;
    #endif
#else
```

```
    typedef char char_t;
#endif

//
// Forward declarations
//

extern void* get_callable_managed_function(
    const char_t* dotnet_type,
    const char_t* dotnet_type_method,
    const char_t* dotnet_delegate_type);

extern void* get_fast_callable_managed_function(
    const char_t* dotnet_type,
    const char_t* dotnet_type_method);

//
// String constants
//

static const char_t* t1_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExports, PF.VsxProtocolDriver.Wrapper");
static const char_t* t2_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer, PF.VsxProtocolDriver.Wrapper");
static const char_t* t3_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsDevice, PF.VsxProtocolDriver.Wrapper");
static const char_t* t4_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsLog, PF.VsxProtocolDriver.Wrapper");
static const char_t* t5_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsParameter, PF.VsxProtocolDriver.Wrapper");
static const char_t* t6_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList, PF.VsxProtocolDriver.Wrapper");
static const char_t* t7_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsResult, PF.VsxProtocolDriver.Wrapper");
static const char_t* t8_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList, PF.VsxProtocolDriver.Wrapper");

//
// Exports
//

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseString
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseString_ptr)(const char** pString);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseString(const char** pString)
{
    if (vsx_ReleaseString_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseString");
        vsx_ReleaseString_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(const char** pString))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ReleaseString_ptr(pString);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetLibraryVersion
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLibraryVersion_ptr)(const char** version);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLibraryVersion(const char** version)
{
    if (vsx_GetLibraryVersion_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetLibraryVersion");
        vsx_GetLibraryVersion_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(const char** version))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_GetLibraryVersion_ptr(version);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetErrorText
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetErrorText_ptr)(int32_t error_code, const char** error_text);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetErrorText(int32_t error_code, const char** error_text)
{
    if (vsx_GetErrorText_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetErrorText");
        vsx_GetErrorText_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(int32_t error_code, const char** error_text))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_GetErrorText_ptr(error_code, error_text);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitTcpSensor
static VsxStatusCode (DNNE_CALLTYPE* vsx_InitTcpSensor_ptr)(VsxSystemHandle** pVsx, const char* ipAddress, const char* pluginName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitTcpSensor(VsxSystemHandle** pVsx, const char* ipAddress, const char* pluginName)
```

```
{
    if (vsx_InitTcpSensor_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("InitTcpSensor");
        vsx_InitTcpSensor_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle** pVsx, const char* ipAddress, const char* pluginName))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_InitTcpSensor_ptr(pVsx, ipAddress, pluginName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitSerialSensor
static VsxStatusCode (DNNE_CALLTYPE* vsx_InitSerialSensor_ptr)(VsxSystemHandle** pVsx, const char* serialPort, int32_t baudrate, const char* sensorType, VsxSerialConnectionType connectionType, const char* pluginName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitSerialSensor(VsxSystemHandle** pVsx, const char* serialPort, int32_t baudrate, const char* sensorType, VsxSerialConnectionType connectionType, const char* pluginName)
{
    if (vsx_InitSerialSensor_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("InitSerialSensor");
        vsx_InitSerialSensor_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle** pVsx, const char* serialPort, int32_t baudrate, const char* sensorType, VsxSerialConnectionType connectionType, const char* pluginName))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_InitSerialSensor_ptr(pVsx, serialPort, baudrate, sensorType, connectionType, pluginName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseSensor
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseSensor_ptr)(VsxSystemHandle** vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseSensor(VsxSystemHandle** vsx)
{
    if (vsx_ReleaseSensor_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseSensor");
        vsx_ReleaseSensor_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle** vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ReleaseSensor_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectTcpDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReConnectTcpDevice_ptr)(VsxSystemHandle* vsx, const char* ipAddress);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectTcpDevice(VsxSystemHandle* vsx, const char* ipAddress)
{
    if (vsx_ReConnectTcpDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReConnectTcpDevice");
        vsx_ReConnectTcpDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* ipAddress))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ReConnectTcpDevice_ptr(vsx, ipAddress);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectAndLoginTcpDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReConnectAndLoginTcpDevice_ptr)(VsxSystemHandle* vsx, const char* ipAddress, const char* username, const char* password);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectAndLoginTcpDevice(VsxSystemHandle* vsx, const char* ipAddress, const char* username, const char* password)
{
    if (vsx_ReConnectAndLoginTcpDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReConnectAndLoginTcpDevice");
        vsx_ReConnectAndLoginTcpDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* ipAddress, const char* username, const char* password))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ReConnectAndLoginTcpDevice_ptr(vsx, ipAddress, username, password);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectSerialDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReConnectSerialDevice_ptr)(VsxSystemHandle* vsx, const char* serialPort, int32_t baudrate, VsxSerialConnectionType connectionType);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectSerialDevice(VsxSystemHandle* vsx, const char* serialPort, int32_t baudrate, VsxSerialConnectionType connectionType)
{
    if (vsx_ReConnectSerialDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReConnectSerialDevice");
        vsx_ReConnectSerialDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* serialPort, int32_t baudrate, VsxSerialConnectionType connectionType))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ReConnectSerialDevice_ptr(vsx, serialPort, baudrate, connectionType);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Connect
static VsxStatusCode (DNNE_CALLTYPE* vsx_Connect_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Connect(VsxSystemHandle* vsx)
{
    if (vsx_Connect_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("Connect");
        vsx_Connect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_Connect_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectEx
static VsxStatusCode (DNNE_CALLTYPE* vsx_ConnectEx_ptr)(VsxSystemHandle* vsx, int32_t timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectEx(VsxSystemHandle* vsx, int32_t timeout_ms)
{
    if (vsx_ConnectEx_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ConnectEx");
        vsx_ConnectEx_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t timeout_ms))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ConnectEx_ptr(vsx, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectAndLogin
static VsxStatusCode (DNNE_CALLTYPE* vsx_ConnectAndLogin_ptr)(VsxSystemHandle* vsx, const char* username, const char* password);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectAndLogin(VsxSystemHandle* vsx, const char* username, const char* password)
{
    if (vsx_ConnectAndLogin_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ConnectAndLogin");
        vsx_ConnectAndLogin_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* username, const char* password))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ConnectAndLogin_ptr(vsx, username, password);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectExAndLogin
static VsxStatusCode (DNNE_CALLTYPE* vsx_ConnectExAndLogin_ptr)(VsxSystemHandle* vsx, const char* username, const char* password, int32_t timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectExAndLogin(VsxSystemHandle* vsx, const char* username, const char* password, int32_t timeout_ms)
{
    if (vsx_ConnectExAndLogin_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ConnectExAndLogin");
        vsx_ConnectExAndLogin_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* username, const char* password, int32_t timeout_ms))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_ConnectExAndLogin_ptr(vsx, username, password, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Login
static VsxStatusCode (DNNE_CALLTYPE* vsx_Login_ptr)(VsxSystemHandle* vsx, const char* username, const char* password);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Login(VsxSystemHandle* vsx, const char* username, const char* password)
{
    if (vsx_Login_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("Login");
        vsx_Login_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* username, const char* password))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_Login_ptr(vsx, username, password);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Logout
static VsxStatusCode (DNNE_CALLTYPE* vsx_Logout_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Logout(VsxSystemHandle* vsx)
{
    if (vsx_Logout_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("Logout");
        vsx_Logout_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_Logout_ptr(vsx);
}
```

```
// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetPassword
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetPassword_ptr)(VsxSystemHandle* vsx, const char* authorizationUsername, const char* authorizationPassword, const char* username, const char* password);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetPassword(VsxSystemHandle* vsx, const char* authorizationUsername, const char* authorizationPassword, const char* username, const char* password)
{
    if (vsx_SetPassword_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetPassword");
        vsx_SetPassword_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* authorizationUsername, const char* authorizationPassword, const char* username, const char* password))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SetPassword_ptr(vsx, authorizationUsername, authorizationPassword, username, password);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetConnected
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetConnected_ptr)(VsxSystemHandle* vsx, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetConnected(VsxSystemHandle* vsx, int32_t* result)
{
    if (vsx_GetConnected_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetConnected");
        vsx_GetConnected_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* result))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_GetConnected_ptr(vsx, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Disconnect
static VsxStatusCode (DNNE_CALLTYPE* vsx_Disconnect_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Disconnect(VsxSystemHandle* vsx)
{
    if (vsx_Disconnect_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("Disconnect");
        vsx_Disconnect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_Disconnect_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnDisconnect
static VsxStatusCode (DNNE_CALLTYPE* vsx_RegisterOnDisconnect_ptr)(VsxSystemHandle* vsx, vsx_OnDisconnect fptr);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDisconnect(VsxSystemHandle* vsx, vsx_OnDisconnect fptr)
{
    if (vsx_RegisterOnDisconnect_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("RegisterOnDisconnect");
        vsx_RegisterOnDisconnect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, vsx_OnDisconnect fptr))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_RegisterOnDisconnect_ptr(vsx, fptr);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnDisconnect
static VsxStatusCode (DNNE_CALLTYPE* vsx_DeregisterOnDisconnect_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDisconnect(VsxSystemHandle* vsx)
{
    if (vsx_DeregisterOnDisconnect_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("DeregisterOnDisconnect");
        vsx_DeregisterOnDisconnect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_DeregisterOnDisconnect_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnSessionMessageReceived
static VsxStatusCode (DNNE_CALLTYPE* vsx_RegisterOnSessionMessageReceived_ptr)(VsxSystemHandle* vsx, vsx_OnSessionMessageReceived fptr);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnSessionMessageReceived(VsxSystemHandle* vsx, vsx_OnSessionMessageReceived fptr)
{
    if (vsx_RegisterOnSessionMessageReceived_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("RegisterOnSessionMessageReceived");
        vsx_RegisterOnSessionMessageReceived_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, vsx_OnSessionMessageReceived fptr))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_RegisterOnSessionMessageReceived_ptr(vsx, fptr);
}

```

```
// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnSessionMessageReceived
static VsxStatusCode (DNNE_CALLTYPE* vsx_DeregisterOnSessionMessageReceived_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnSessionMessageReceived(VsxSystemHandle* vsx)
{
    if (vsx_DeregisterOnSessionMessageReceived_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("DeregisterOnSessionMessageReceived");
        vsx_DeregisterOnSessionMessageReceived_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_DeregisterOnSessionMessageReceived_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendSessionKeepAlive
static VsxStatusCode (DNNE_CALLTYPE* vsx_SendSessionKeepAlive_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendSessionKeepAlive(VsxSystemHandle* vsx)
{
    if (vsx_SendSessionKeepAlive_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SendSessionKeepAlive");
        vsx_SendSessionKeepAlive_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SendSessionKeepAlive_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystem
static VsxStatusCode (DNNE_CALLTYPE* vsx_TestSystem_ptr)(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystem(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status)
{
    if (vsx_TestSystem_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("TestSystem");
        vsx_TestSystem_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_TestSystem_ptr(vsx, command, inputValue, outputValue, status);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystemEx
static VsxStatusCode (DNNE_CALLTYPE* vsx_TestSystemEx_ptr)(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status, int32_t timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystemEx(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status, int32_t timeout_ms)
{
    if (vsx_TestSystemEx_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("TestSystemEx");
        vsx_TestSystemEx_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* command, const char* inputValue, const char** outputValue, int32_t* status, int32_t timeout_ms))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_TestSystemEx_ptr(vsx, command, inputValue, outputValue, status, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetWaitTimeout
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetWaitTimeout_ptr)(VsxSystemHandle* vsx, int32_t* timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetWaitTimeout(VsxSystemHandle* vsx, int32_t* timeout_ms)
{
    if (vsx_GetWaitTimeout_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetWaitTimeout");
        vsx_GetWaitTimeout_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* timeout_ms))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_GetWaitTimeout_ptr(vsx, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetWaitTimeout
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetWaitTimeout_ptr)(VsxSystemHandle* vsx, int32_t timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetWaitTimeout(VsxSystemHandle* vsx, int32_t timeout_ms)
{
    if (vsx_SetWaitTimeout_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetWaitTimeout");
        vsx_SetWaitTimeout_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t timeout_ms))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SetWaitTimeout_ptr(vsx, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.UploadData
static VsxStatusCode (DNNE_CALLTYPE* vsx_UploadData_ptr)(VsxSystemHandle* vsx, const char* fileName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadData(VsxSystemHandle* vsx, const char* fileName)
{
    if (vsx_UploadData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("UploadData");
        vsx_UploadData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* fileName))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_UploadData_ptr(vsx, fileName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendFirmware
static VsxStatusCode (DNNE_CALLTYPE* vsx_SendFirmware_ptr)(VsxSystemHandle* vsx, const char* fileName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendFirmware(VsxSystemHandle* vsx, const char* fileName)
{
    if (vsx_SendFirmware_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SendFirmware");
        vsx_SendFirmware_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* fileName))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SendFirmware_ptr(vsx, fileName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendXmlDataMessage
static VsxStatusCode (DNNE_CALLTYPE* vsx_SendXmlDataMessage_ptr)(VsxSystemHandle* vsx, const char* xmlCommand);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendXmlDataMessage(VsxSystemHandle* vsx, const char* xmlCommand)
{
    if (vsx_SendXmlDataMessage_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SendXmlDataMessage");
        vsx_SendXmlDataMessage_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* xmlCommand))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SendXmlDataMessage_ptr(vsx, xmlCommand);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettings
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetNetworkSettings_ptr)(VsxSystemHandle* vsx, const char* ipAddress, const char* networkMask, const char* gateway);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettings(VsxSystemHandle* vsx, const char* ipAddress, const char* networkMask, const char* gateway)
{
    if (vsx_SetNetworkSettings_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetNetworkSettings");
        vsx_SetNetworkSettings_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* ipAddress, const char* networkMask, const char* gateway))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SetNetworkSettings_ptr(vsx, ipAddress, networkMask, gateway);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettingsViaUdp
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetNetworkSettingsViaUdp_ptr)(const char* macAddress, const char* ipAddress, const char* networkMask, const char* gateway);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettingsViaUdp(const char* macAddress, const char* ipAddress, const char* networkMask, const char* gateway)
{
    if (vsx_SetNetworkSettingsViaUdp_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetNetworkSettingsViaUdp");
        vsx_SetNetworkSettingsViaUdp_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(const char* macAddress, const char* ipAddress, const char* networkMask, const char* gateway))get_fast_callable_managed_function(t1_name, methodName);
    }
    return vsx_SetNetworkSettingsViaUdp_ptr(macAddress, ipAddress, networkMask, gateway);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ResetDynamicContainerGrabber
static VsxStatusCode (DNNE_CALLTYPE* vsx_ResetDynamicContainerGrabber_ptr)(VsxSystemHandle* vsx, int32_t bufferSize, VsxStrategy strategy);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetDynamicContainerGrabber(VsxSystemHandle* vsx, int32_t bufferSize, VsxStrategy strategy)
{
    if (vsx_ResetDynamicContainerGrabber_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ResetDynamicContainerGrabber");
        vsx_ResetDynamicContainerGrabber_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t bufferSize, VsxStrategy strategy))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ResetDynamicContainerGrabber_ptr(vsx, bufferSize, strategy);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDataContainer
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDataContainer_ptr)(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDataContainer(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t timeout_ms)
{
    if (vsx_GetDataContainer_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetDataContainer");
        vsx_GetDataContainer_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t timeout_ms))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetDataContainer_ptr(vsx, pDch, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCachedContainer
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetCachedContainer_ptr)(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t position);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCachedContainer(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t position)
{
    if (vsx_GetCachedContainer_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetCachedContainer");
        vsx_GetCachedContainer_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, VsxDataContainerHandle** pDch, int32_t position))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetCachedContainer_ptr(vsx, pDch, position);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDataContainer
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDataContainer_ptr)(VsxDataContainerHandle** dch);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDataContainer(VsxDataContainerHandle** dch)
{
    if (vsx_ReleaseDataContainer_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseDataContainer");
        vsx_ReleaseDataContainer_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle** dch))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseDataContainer_ptr(dch);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.SaveData
static VsxStatusCode (DNNE_CALLTYPE* vsx_SaveData_ptr)(VsxDataContainerHandle* dch, const char* tag, const char* fileName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveData(VsxDataContainerHandle* dch, const char* tag, const char* fileName)
{
    if (vsx_SaveData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SaveData");
        vsx_SaveData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, const char* fileName))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_SaveData_ptr(dch, tag, fileName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.Save3DPointCloudData
static VsxStatusCode (DNNE_CALLTYPE* vsx_Save3DPointCloudData_ptr)(VsxDataContainerHandle* dch, const char* point_x_Id, const char* point_y_Id, const char* point_z_Id, const char* fileName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Save3DPointCloudData(VsxDataContainerHandle* dch, const char* point_x_Id, const char* point_y_Id, const char* point_z_Id, const char* fileName)
{
    if (vsx_Save3DPointCloudData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("Save3DPointCloudData");
        vsx_Save3DPointCloudData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* point_x_Id, const char* point_y_Id, const char* point_z_Id, const char* fileName))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_Save3DPointCloudData_ptr(dch, point_x_Id, point_y_Id, point_z_Id, fileName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetImage
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetImage_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxImage** imageData);
```

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetImage(VsxDataContainerHandle* dch, const char* tag, VsxImage** imageData)
{
    if (vsx_GetImage_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetImage");
        vsx_GetImage_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxImage** imageData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetImage_ptr(dch, tag, imageData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseImage
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseImage_ptr)(VsxImage** pImage);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseImage(VsxImage** pImage)
{
    if (vsx_ReleaseImage_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseImage");
        vsx_ReleaseImage_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxImage** pImage))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseImage_ptr(pImage);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetLine
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLine_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxLineData** data);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLine(VsxDataContainerHandle* dch, const char* tag, VsxLineData** data)
{
    if (vsx_GetLine_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetLine");
        vsx_GetLine_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxLineData** data))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetLine_ptr(dch, tag, data);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseLine
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseLine_ptr)(VsxLineData** pLineData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseLine(VsxLineData** pLineData)
{
    if (vsx_ReleaseLine_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseLine");
        vsx_ReleaseLine_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxLineData** pLineData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseLine_ptr(pLineData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDisparityDescriptor2
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDisparityDescriptor2_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxDisparityDescriptor2** data);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2(VsxDataContainerHandle* dch, const char* tag, VsxDisparityDescriptor2** data)
{
    if (vsx_GetDisparityDescriptor2_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetDisparityDescriptor2");
        vsx_GetDisparityDescriptor2_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxDisparityDescriptor2** data))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetDisparityDescriptor2_ptr(dch, tag, data);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDisparityDescriptor2
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDisparityDescriptor2_ptr)(VsxDisparityDescriptor2** pData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDisparityDescriptor2(VsxDisparityDescriptor2** pData)
{
    if (vsx_ReleaseDisparityDescriptor2_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseDisparityDescriptor2");
        vsx_ReleaseDisparityDescriptor2_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDisparityDescriptor2** pData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseDisparityDescriptor2_ptr(pData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTransformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetTransformation_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxTransformation** data);
```

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTransformation(VsxDataContainerHandle* dch, const char* tag, VsxTransformation** data)
{
    if (vsx_GetTransformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetTransformation");
        vsx_GetTransformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxTransformation** data))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetTransformation_ptr(dch, tag, data);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTransformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseTransformation_ptr)(VsxTransformation** pData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTransformation(VsxTransformation** pData)
{
    if (vsx_ReleaseTransformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseTransformation");
        vsx_ReleaseTransformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxTransformation** pData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseTransformation_ptr(pData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCaptureInformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetCaptureInformation_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxCaptureInformation** data);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCaptureInformation(VsxDataContainerHandle* dch, const char* tag, VsxCaptureInformation** data)
{
    if (vsx_GetCaptureInformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetCaptureInformation");
        vsx_GetCaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxCaptureInformation** data))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetCaptureInformation_ptr(dch, tag, data);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseCaptureInformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseCaptureInformation_ptr)(VsxCaptureInformation** pData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseCaptureInformation(VsxCaptureInformation** pData)
{
    if (vsx_ReleaseCaptureInformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseCaptureInformation");
        vsx_ReleaseCaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxCaptureInformation** pData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseCaptureInformation_ptr(pData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2CaptureInformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetOlr2CaptureInformation_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxOlr2CaptureInformation** data);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2CaptureInformation(VsxDataContainerHandle* dch, const char* tag, VsxOlr2CaptureInformation** data)
{
    if (vsx_GetOlr2CaptureInformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetOlr2CaptureInformation");
        vsx_GetOlr2CaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxOlr2CaptureInformation** data))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetOlr2CaptureInformation_ptr(dch, tag, data);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2CaptureInformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseOlr2CaptureInformation_ptr)(VsxOlr2CaptureInformation** pData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2CaptureInformation(VsxOlr2CaptureInformation** pData)
{
    if (vsx_ReleaseOlr2CaptureInformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseOlr2CaptureInformation");
        vsx_ReleaseOlr2CaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxOlr2CaptureInformation** pData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseOlr2CaptureInformation_ptr(pData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2ModbusData
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetOlr2ModbusData_ptr)(VsxDataContainerHandle* dch, const char* tag, VsxOlr2ModbusData** data);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2ModbusData(VsxDataContainerHandle* dch, const char* tag, VsxOlr2ModbusData** data)
{
    if (vsx_GetOlr2ModbusData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetOlr2ModbusData");
        vsx_GetOlr2ModbusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* tag, VsxOlr2ModbusData** data))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetOlr2ModbusData_ptr(dch, tag, data);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2ModbusData
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseOlr2ModbusData_ptr)(VsxOlr2ModbusData** pData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2ModbusData(VsxOlr2ModbusData** pData)
{
    if (vsx_ReleaseOlr2ModbusData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseOlr2ModbusData");
        vsx_ReleaseOlr2ModbusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxOlr2ModbusData** pData))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseOlr2ModbusData_ptr(pData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTagList
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetTagList_ptr)(VsxDataContainerHandle* dch, VsxTagList** tagList);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTagList(VsxDataContainerHandle* dch, VsxTagList** tagList)
{
    if (vsx_GetTagList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetTagList");
        vsx_GetTagList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, VsxTagList** tagList))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetTagList_ptr(dch, tagList);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTagList
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseTagList_ptr)(VsxTagList** pTagList);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTagList(VsxTagList** pTagList)
{
    if (vsx_ReleaseTagList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseTagList");
        vsx_ReleaseTagList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxTagList** pTagList))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_ReleaseTagList_ptr(pTagList);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetMissingContainerFramesCounter
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetMissingContainerFramesCounter_ptr)(VsxSystemHandle* vsx, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingContainerFramesCounter(VsxSystemHandle* vsx, int32_t* result)
{
    if (vsx_GetMissingContainerFramesCounter_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetMissingContainerFramesCounter");
        vsx_GetMissingContainerFramesCounter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* result))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetMissingContainerFramesCounter_ptr(vsx, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDynamicContainerQueueSize
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDynamicContainerQueueSize_ptr)(VsxSystemHandle* vsx, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDynamicContainerQueueSize(VsxSystemHandle* vsx, int32_t* result)
{
    if (vsx_GetDynamicContainerQueueSize_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetDynamicContainerQueueSize");
        vsx_GetDynamicContainerQueueSize_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* result))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetDynamicContainerQueueSize_ptr(vsx, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetNumberOfCachedContainers
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetNumberOfCachedContainers_ptr)(VsxSystemHandle* vsx, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetNumberOfCachedContainers(VsxSystemHandle* vsx, int32_t* result)
{
    if (vsx_GetNumberOfCachedContainers_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetNumberOfCachedContainers");
        vsx_GetNumberOfCachedContainers_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* result))get_fast_callable_managed_function(t2_name, methodName);
    }
    return vsx_GetNumberOfCachedContainers_ptr(vsx, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetDeviceInformation
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDeviceInformation_ptr)(VsxSystemHandle* vsx, VsxDevice** deviceData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDeviceInformation(VsxSystemHandle* vsx, VsxDevice** deviceData)
{
    if (vsx_GetDeviceInformation_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetDeviceInformation");
        vsx_GetDeviceInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, VsxDevice** deviceData))get_fast_callable_managed_function(t3_name, methodName);
    }
    return vsx_GetDeviceInformation_ptr(vsx, deviceData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDevice_ptr)(VsxDevice** pDevice);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDevice(VsxDevice** pDevice)
{
    if (vsx_ReleaseDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseDevice");
        vsx_ReleaseDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDevice** pDevice))get_fast_callable_managed_function(t3_name, methodName);
    }
    return vsx_ReleaseDevice_ptr(pDevice);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetUdpDeviceList
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetUdpDeviceList_ptr)(VsxDeviceList** deviceListData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetUdpDeviceList(VsxDeviceList** deviceListData)
{
    if (vsx_GetUdpDeviceList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetUdpDeviceList");
        vsx_GetUdpDeviceList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDeviceList** deviceListData))get_fast_callable_managed_function(t3_name, methodName);
    }
    return vsx_GetUdpDeviceList_ptr(deviceListData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDeviceList
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDeviceList_ptr)(VsxDeviceList** pDeviceList);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDeviceList(VsxDeviceList** pDeviceList)
{
    if (vsx_ReleaseDeviceList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseDeviceList");
        vsx_ReleaseDeviceList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDeviceList** pDeviceList))get_fast_callable_managed_function(t3_name, methodName);
    }
    return vsx_ReleaseDeviceList_ptr(pDeviceList);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.ResetLogMessageGrabber
static VsxStatusCode (DNNE_CALLTYPE* vsx_ResetLogMessageGrabber_ptr)(VsxSystemHandle* vsx, int32_t bufferSize, int32_t typeMask, VsxStrategy strategy);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetLogMessageGrabber(VsxSystemHandle* vsx, int32_t bufferSize, int32_t typeMask, VsxStrategy strategy)
{
    if (vsx_ResetLogMessageGrabber_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ResetLogMessageGrabber");
        vsx_ResetLogMessageGrabber_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t bufferSize, int32_t typeMask, VsxStrategy strategy))get_fast_callable_managed_function(t4_name, methodName);
    }
    return vsx_ResetLogMessageGrabber_ptr(vsx, bufferSize, typeMask, strategy);
```

```
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessage
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLogMessage_ptr)(VsxSystemHandle* vsx, const char** log, int32_t timeout_ms);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessage(VsxSystemHandle* vsx, const char** log, int32_t timeout_ms)
{
    if (vsx_GetLogMessage_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetLogMessage");
        vsx_GetLogMessage_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char** log, int32_t timeout_ms))get_fast_callable_managed_function(t4_name, methodName);
    }
    return vsx_GetLogMessage_ptr(vsx, log, timeout_ms);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessageQueueSize
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLogMessageQueueSize_ptr)(VsxSystemHandle* vsx, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessageQueueSize(VsxSystemHandle* vsx, int32_t* result)
{
    if (vsx_GetLogMessageQueueSize_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetLogMessageQueueSize");
        vsx_GetLogMessageQueueSize_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* result))get_fast_callable_managed_function(t4_name, methodName);
    }
    return vsx_GetLogMessageQueueSize_ptr(vsx, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetMissingLogMessagesCounter
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetMissingLogMessagesCounter_ptr)(VsxSystemHandle* vsx, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingLogMessagesCounter(VsxSystemHandle* vsx, int32_t* result)
{
    if (vsx_GetMissingLogMessagesCounter_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetMissingLogMessagesCounter");
        vsx_GetMissingLogMessagesCounter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, int32_t* result))get_fast_callable_managed_function(t4_name, methodName);
    }
    return vsx_GetMissingLogMessagesCounter_ptr(vsx, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValue
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterValue_ptr)(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, const char* value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValue(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, const char* value)
{
    if (vsx_SetSingleParameterValue_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetSingleParameterValue");
        vsx_SetSingleParameterValue_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, const char* value))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_SetSingleParameterValue_ptr(vsx, settingsVersion, configurationId, configurationVersion, parameterId, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueDouble
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterValueDouble_ptr)(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, double value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueDouble(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, double value)
{
    if (vsx_SetSingleParameterValueDouble_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetSingleParameterValueDouble");
        vsx_SetSingleParameterValueDouble_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId, double value))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_SetSingleParameterValueDouble_ptr(vsx, settingsVersion, configurationId, configurationVersion, parameterId, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueInt32
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterValueInt32_ptr)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterValueInt32(VsxSystemHandle*
vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t value)
{
    if (vsx_SetSingleParameterValueInt32_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetSingleParameterValueInt32");
        vsx_SetSingleParameterValueInt32_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t value))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_SetSingleParameterValueInt32_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValue
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameterValue_ptr)(VsxSystemHandle* vsx, uint32_t
settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId,
const char** value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValue(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, const char** value)
{
    if (vsx_GetSingleParameterValue_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetSingleParameterValue");
        vsx_GetSingleParameterValue_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, const char** value))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_GetSingleParameterValue_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueDouble
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameterValueDouble_ptr)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, double* value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueDouble(VsxSystemHandle*
vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, double* value)
{
    if (vsx_GetSingleParameterValueDouble_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetSingleParameterValueDouble");
        vsx_GetSingleParameterValueDouble_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, double* value))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_GetSingleParameterValueDouble_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueInt32
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameterValueInt32_ptr)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t* value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterValueInt32(VsxSystemHandle*
vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t* value)
{
    if (vsx_GetSingleParameterValueInt32_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetSingleParameterValueInt32");
        vsx_GetSingleParameterValueInt32_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t* value))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_GetSingleParameterValueInt32_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadDefaultParameterSetOnDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_LoadDefaultParameterSetOnDevice_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_LoadDefaultParameterSetOnDevice(VsxSystemHandle* vsx)
{
    if (vsx_LoadDefaultParameterSetOnDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("LoadDefaultParameterSetOnDevice");
        vsx_LoadDefaultParameterSetOnDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle*
vsx))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_LoadDefaultParameterSetOnDevice_ptr(vsx);
}
```

```

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadParameterSetOnDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_LoadParameterSetOnDevice_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadParameterSetOnDevice(VsxSystemHandle* vsx)
{
    if (vsx_LoadParameterSetOnDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("LoadParameterSetOnDevice");
        vsx_LoadParameterSetOnDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_LoadParameterSetOnDevice_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SaveParameterSetOnDevice
static VsxStatusCode (DNNE_CALLTYPE* vsx_SaveParameterSetOnDevice_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveParameterSetOnDevice(VsxSystemHandle* vsx)
{
    if (vsx_SaveParameterSetOnDevice_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SaveParameterSetOnDevice");
        vsx_SaveParameterSetOnDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_SaveParameterSetOnDevice_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.UploadParameterSet
static VsxStatusCode (DNNE_CALLTYPE* vsx_UploadParameterSet_ptr)(VsxSystemHandle* vsx, const char* fileName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterSet(VsxSystemHandle* vsx, const char* fileName)
{
    if (vsx_UploadParameterSet_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("UploadParameterSet");
        vsx_UploadParameterSet_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* fileName))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_UploadParameterSet_ptr(vsx, fileName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.DownloadParameterSet
static VsxStatusCode (DNNE_CALLTYPE* vsx_DownloadParameterSet_ptr)(VsxSystemHandle* vsx, const char* fileName);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DownloadParameterSet(VsxSystemHandle* vsx, const char* fileName)
{
    if (vsx_DownloadParameterSet_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("DownloadParameterSet");
        vsx_DownloadParameterSet_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const char* fileName))get_fast_callable_managed_function(t5_name, methodName);
    }
    return vsx_DownloadParameterSet_ptr(vsx, fileName);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetParameterList
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetParameterList_ptr)(VsxSystemHandle* vsx, VsxParameterList** parameterListData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetParameterList(VsxSystemHandle* vsx, VsxParameterList** parameterListData)
{
    if (vsx_GetParameterList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetParameterList");
        vsx_GetParameterList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, VsxParameterList** parameterListData))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_GetParameterList_ptr(vsx, parameterListData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.UploadParameterList
static VsxStatusCode (DNNE_CALLTYPE* vsx_UploadParameterList_ptr)(VsxSystemHandle* vsx, VsxParameterList* parameterListData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterList(VsxSystemHandle* vsx, VsxParameterList* parameterListData)
{
    if (vsx_UploadParameterList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("UploadParameterList");
        vsx_UploadParameterList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, VsxParameterList* parameterListData))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_UploadParameterList_ptr(vsx, parameterListData);
}

```

```
// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterString
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterString_ptr)(VsxSystemHandle* vsx, const VsxParameter* parameter, const char* value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterString(VsxSystemHandle* vsx, const VsxParameter* parameter, const char* value)
{
    if (vsx_SetSingleParameterString_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetSingleParameterString");
        vsx_SetSingleParameterString_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const VsxParameter* parameter, const char* value))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_SetSingleParameterString_ptr(vsx, parameter, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterDouble
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterDouble_ptr)(VsxSystemHandle* vsx, const VsxParameter* parameter, double value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble(VsxSystemHandle* vsx, const VsxParameter* parameter, double value)
{
    if (vsx_SetSingleParameterDouble_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetSingleParameterDouble");
        vsx_SetSingleParameterDouble_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const VsxParameter* parameter, double value))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_SetSingleParameterDouble_ptr(vsx, parameter, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterInt32
static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterInt32_ptr)(VsxSystemHandle* vsx, const VsxParameter* parameter, int32_t value);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterInt32(VsxSystemHandle* vsx, const VsxParameter* parameter, int32_t value)
{
    if (vsx_SetSingleParameterInt32_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SetSingleParameterInt32");
        vsx_SetSingleParameterInt32_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const VsxParameter* parameter, int32_t value))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_SetSingleParameterInt32_ptr(vsx, parameter, value);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetSingleParameter
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameter_ptr)(VsxSystemHandle* vsx, const VsxParameter* parameterIn, const VsxParameter** parameterOut);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameter(VsxSystemHandle* vsx, const VsxParameter* parameterIn, const VsxParameter** parameterOut)
{
    if (vsx_GetSingleParameter_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetSingleParameter");
        vsx_GetSingleParameter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, const VsxParameter* parameterIn, const VsxParameter** parameterOut))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_GetSingleParameter_ptr(vsx, parameterIn, parameterOut);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameter
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseParameter_ptr)(const VsxParameter** pParameter);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameter(const VsxParameter** pParameter)
{
    if (vsx_ReleaseParameter_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseParameter");
        vsx_ReleaseParameter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(const VsxParameter** pParameter))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_ReleaseParameter_ptr(pParameter);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameterList
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseParameterList_ptr)(VsxParameterList** pParameterList);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameterList(VsxParameterList** pParameterList)
{
    if (vsx_ReleaseParameterList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseParameterList");
        vsx_ReleaseParameterList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxParameterList** pParameterList))get_fast_callable_managed_function(t6_name, methodName);
    }
    return vsx_ReleaseParameterList_ptr(pParameterList);
```

```
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultXml
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultXml_ptr)(VsxDataContainerHandle* dch, const char* resultId, const char** result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultXml(VsxDataContainerHandle* dch, const char* resultId, const char** result)
{
    if (vsx_GetResultXml_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetResultXml");
        vsx_GetResultXml_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* resultId, const char** result))get_fast_callable_managed_function(t7_name, methodName);
    }
    return vsx_GetResultXml_ptr(dch, resultId, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementString
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementString_ptr)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, const char** result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementString(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, const char** result)
{
    if (vsx_GetResultElementString_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetResultElementString");
        vsx_GetResultElementString_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, const char** result))get_fast_callable_managed_function(t7_name, methodName);
    }
    return vsx_GetResultElementString_ptr(dch, resultId, xPath, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt32
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementInt32_ptr)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, int32_t* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, int32_t* result)
{
    if (vsx_GetResultElementInt32_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetResultElementInt32");
        vsx_GetResultElementInt32_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, int32_t* result))get_fast_callable_managed_function(t7_name, methodName);
    }
    return vsx_GetResultElementInt32_ptr(dch, resultId, xPath, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt64
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementInt64_ptr)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, LOCAL_INT64_T* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt64(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, LOCAL_INT64_T* result)
{
    if (vsx_GetResultElementInt64_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetResultElementInt64");
        vsx_GetResultElementInt64_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, LOCAL_INT64_T* result))get_fast_callable_managed_function(t7_name, methodName);
    }
    return vsx_GetResultElementInt64_ptr(dch, resultId, xPath, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementDouble
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementDouble_ptr)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, double* result);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementDouble(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, double* result)
{
    if (vsx_GetResultElementDouble_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetResultElementDouble");
        vsx_GetResultElementDouble_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxDataContainerHandle* dch, const char* resultId, const char* xPath, double* result))get_fast_callable_managed_function(t7_name, methodName);
    }
    return vsx_GetResultElementDouble_ptr(dch, resultId, xPath, result);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.GetAllDeviceStatusData
static VsxStatusCode (DNNE_CALLTYPE* vsx_GetAllDeviceStatusData_ptr)(VsxSystemHandle* vsx, VsxStatusItemList** statusItemListData);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetAllDeviceStatusData(VsxSystemHandle* vsx, VsxStatusItemList** statusItemListData)
{
    if (vsx_GetAllDeviceStatusData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("GetAllDeviceStatusData");
        vsx_GetAllDeviceStatusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, VsxStatusItemList** statusItemListData))get_fast_callable_managed_function(t8_name, methodName);
    }
    return vsx_GetAllDeviceStatusData_ptr(vsx, statusItemListData);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.ReleaseStatusItemList
static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseStatusItemList_ptr)(VsxStatusItemList** pStatusItemList);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseStatusItemList(VsxStatusItemList** pStatusItemList)
{
    if (vsx_ReleaseStatusItemList_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("ReleaseStatusItemList");
        vsx_ReleaseStatusItemList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxStatusItemList** pStatusItemList))get_fast_callable_managed_function(t8_name, methodName);
    }
    return vsx_ReleaseStatusItemList_ptr(pStatusItemList);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.RegisterOnDeviceStatusReceived
static VsxStatusCode (DNNE_CALLTYPE* vsx_RegisterOnDeviceStatusReceived_ptr)(VsxSystemHandle* vsx, vsx_OnDeviceStatusReceived fptr);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDeviceStatusReceived(VsxSystemHandle* vsx, vsx_OnDeviceStatusReceived fptr)
{
    if (vsx_RegisterOnDeviceStatusReceived_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("RegisterOnDeviceStatusReceived");
        vsx_RegisterOnDeviceStatusReceived_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx, vsx_OnDeviceStatusReceived fptr))get_fast_callable_managed_function(t8_name, methodName);
    }
    return vsx_RegisterOnDeviceStatusReceived_ptr(vsx, fptr);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.DeregisterOnDeviceStatusReceived
static VsxStatusCode (DNNE_CALLTYPE* vsx_DeregisterOnDeviceStatusReceived_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDeviceStatusReceived(VsxSystemHandle* vsx)
{
    if (vsx_DeregisterOnDeviceStatusReceived_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("DeregisterOnDeviceStatusReceived");
        vsx_DeregisterOnDeviceStatusReceived_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t8_name, methodName);
    }
    return vsx_DeregisterOnDeviceStatusReceived_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.SubscribeToDeviceStatusData
static VsxStatusCode (DNNE_CALLTYPE* vsx_SubscribeToDeviceStatusData_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SubscribeToDeviceStatusData(VsxSystemHandle* vsx)
{
    if (vsx_SubscribeToDeviceStatusData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("SubscribeToDeviceStatusData");
        vsx_SubscribeToDeviceStatusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t8_name, methodName);
    }
    return vsx_SubscribeToDeviceStatusData_ptr(vsx);
}

// Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.UnsubscribeToDeviceStatusData
static VsxStatusCode (DNNE_CALLTYPE* vsx_UnsubscribeToDeviceStatusData_ptr)(VsxSystemHandle* vsx);
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UnsubscribeToDeviceStatusData(VsxSystemHandle* vsx)
{
    if (vsx_UnsubscribeToDeviceStatusData_ptr == NULL)
    {
        const char_t* methodName = DNNE_STR("UnsubscribeToDeviceStatusData");
        vsx_UnsubscribeToDeviceStatusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle* vsx))get_fast_callable_managed_function(t8_name, methodName);
    }
    return vsx_UnsubscribeToDeviceStatusData_ptr(vsx);
}

#endif // DNNE_COMPILE_AS_SOURCE
```

# Index