



## **VsxProtocolDriver C-Wrapper**

3.3.2+ge9b14e0

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

<b>1 Introduction</b>	<b>1</b>
1.1 Supported devices . . . . .	1
1.2 Requirements . . . . .	2
<b>2 Usage with C interface</b>	<b>2</b>
2.1 Requirements . . . . .	2
2.2 Installation . . . . .	2
2.3 Documentation . . . . .	3
2.3.1 Memory management . . . . .	3
2.3.2 Error handling . . . . .	3
2.3.3 Support of dynamic container (including 3D data) . . . . .	4
2.3.4 Firmware Update . . . . .	4
2.4 Functions . . . . .	5
2.4.1 Common library functions . . . . .	5
2.4.2 Sensor handling . . . . .	5
2.4.3 Sensor handling (Reconnect functions) . . . . .	5
2.4.4 Sensor functions (excluding dynamic container) . . . . .	5
2.4.5 Log message handling . . . . .	6
2.4.6 Dynamic container handling . . . . .	7
2.4.7 Display device information . . . . .	8
<b>3 Examples</b>	<b>8</b>
<b>4 Device parameter</b>	<b>9</b>
<b>5 Changelog</b>	<b>10</b>
<b>6 Class Index</b>	<b>12</b>
6.1 Class List . . . . .	12
<b>7 File Index</b>	<b>13</b>
7.1 File List . . . . .	13
<b>8 Class Documentation</b>	<b>13</b>
8.1 _VsxCaptureInformation Struct Reference . . . . .	13
8.1.1 Detailed Description . . . . .	14
8.1.2 Member Data Documentation . . . . .	14
8.2 _VsxDatContainerHandle Struct Reference . . . . .	15
8.2.1 Detailed Description . . . . .	15
8.2.2 Member Data Documentation . . . . .	15
8.3 _VsxDDevice Struct Reference . . . . .	15
8.3.1 Detailed Description . . . . .	16
8.3.2 Member Data Documentation . . . . .	16
8.4 _VsxDDeviceList Struct Reference . . . . .	17
8.4.1 Detailed Description . . . . .	18

8.4.2 Member Data Documentation . . . . .	18
8.5 _VsxDisparityDescriptor2 Struct Reference . . . . .	18
8.5.1 Detailed Description . . . . .	18
8.5.2 Member Data Documentation . . . . .	18
8.6 _VsxImage Struct Reference . . . . .	19
8.6.1 Detailed Description . . . . .	19
8.6.2 Member Data Documentation . . . . .	19
8.7 _VsxLineCoordinate Struct Reference . . . . .	21
8.7.1 Detailed Description . . . . .	21
8.7.2 Member Data Documentation . . . . .	21
8.8 _VsxLineData Struct Reference . . . . .	22
8.8.1 Detailed Description . . . . .	22
8.8.2 Member Data Documentation . . . . .	22
8.9 _VsxOlr2CaptureInformation Struct Reference . . . . .	23
8.9.1 Detailed Description . . . . .	24
8.9.2 Member Data Documentation . . . . .	24
8.10 _VsxOlr2ModbusData Struct Reference . . . . .	26
8.10.1 Detailed Description . . . . .	26
8.10.2 Member Data Documentation . . . . .	26
8.11 _VsxParameter Struct Reference . . . . .	27
8.11.1 Detailed Description . . . . .	27
8.11.2 Member Data Documentation . . . . .	27
8.12 _VsxParameterEnumItem Struct Reference . . . . .	28
8.12.1 Detailed Description . . . . .	29
8.12.2 Member Data Documentation . . . . .	29
8.13 _VsxParameterList Struct Reference . . . . .	29
8.13.1 Detailed Description . . . . .	29
8.13.2 Member Data Documentation . . . . .	29
8.14 _VsxStatusItem Struct Reference . . . . .	29
8.14.1 Detailed Description . . . . .	30
8.14.2 Member Data Documentation . . . . .	30
8.15 _VsxStatusItemList Struct Reference . . . . .	31
8.15.1 Detailed Description . . . . .	31
8.15.2 Member Data Documentation . . . . .	31
8.16 _VsxSystemHandle Struct Reference . . . . .	32
8.16.1 Detailed Description . . . . .	32
8.16.2 Member Data Documentation . . . . .	32
8.17 _VsxTagList Struct Reference . . . . .	32
8.17.1 Detailed Description . . . . .	32
8.17.2 Member Data Documentation . . . . .	33
8.18 _VsxTransformation Struct Reference . . . . .	33
8.18.1 Detailed Description . . . . .	33

8.18.2 Member Data Documentation . . . . .	33
<b>9 File Documentation</b>	<b>34</b>
9.1 01_Introduction.md File Reference . . . . .	34
9.2 02_Usage_with_c.md File Reference . . . . .	34
9.3 03_Examples.md File Reference . . . . .	34
9.4 04_Device_parameter.md File Reference . . . . .	34
9.5 05_Changelog.md File Reference . . . . .	34
9.6 dnne.h File Reference . . . . .	34
9.6.1 Macro Definition Documentation . . . . .	35
9.6.2 Typedef Documentation . . . . .	36
9.6.3 Enumeration Type Documentation . . . . .	36
9.6.4 Function Documentation . . . . .	36
9.7 dnne.h . . . . .	37
9.8 PF.VsxProtocolDriver.WrapperNE.h File Reference . . . . .	38
9.8.1 Macro Definition Documentation . . . . .	49
9.8.2 Typedef Documentation . . . . .	49
9.8.3 Enumeration Type Documentation . . . . .	53
9.8.4 Function Documentation . . . . .	60
9.9 PF.VsxProtocolDriver.WrapperNE.h . . . . .	99
<b>Index</b>	<b>127</b>

# 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 probably 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 probably 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
<code>vsx_ReleaseString</code>	<code>const char*</code>
<code>vsx_ReleaseSensor</code>	<code>VsxSystemHandle*</code>
<code>vsx_ReleaseDataContainer</code>	<code>VsxDataContainerHandle*</code>
<code>vsx_ReleaseImage</code>	<code>VsxImage*</code>
<code>vsx_ReleaseLine</code>	<code>VsxLine*</code>
<code>vsx_ReleaseDisparityDescriptor2</code>	<code>VsxDisparityDescriptor2*</code>
<code>vsx_ReleaseTransformation</code>	<code>VsxTransformation*</code>
<code>vsx_ReleaseCaptureInformation</code>	<code>VsxCaptureInformation*</code>
<code>vsx_ReleaseOlr2CaptureInformation</code>	<code>VsxOlr2CaptureInformation*</code>
<code>vsx_ReleaseOlr2ModbusData</code>	<code>VsxOlr2ModbusData*</code>
<code>vsx_ReleaseTagList</code>	<code>VsxTagList*</code>
<code>vsx_ReleaseDevice</code>	<code>VsxDevice*</code>
<code>vsx_ReleaseDeviceList</code>	<code>VsxDeviceList*</code>
<code>vsx_ReleaseParameter</code>	<code>VsxParameter*</code>
<code>vsx_ReleaseParameterList</code>	<code>VsxParameterList*</code>
<code>vsx_ReleaseStatusItemList</code>	<code>VsxStatusItemList*</code>

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 following 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 behaviour 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`.

```
// allocate 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 afterwards.
  - 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 management 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 VsxCStatusCode (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, VsxCSerialConnectionType 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,
VsxCSerialConnectionType 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 outputValue 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 individual 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 available.

```
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 structure has two layers. 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 possibility 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 sensor, which configurations are available  
configurationId -> Name of current configuration group  
configurationVersion -> Version number, which tells the sensor, 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 serialization will guarantee also in case of a 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 mask 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. Therefore `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)
```

Sometimes 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 within 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 Smarrunner 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 documentation 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\_SetSingleParameterString"
- 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 "VsxParameterValue" & "VsxStatusItemValue"

## 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:

<a href="#"><u>_VsxCaptureInformation</u></a>	
Contains information about image capture	13
<a href="#"><u>_VsxDataContainerHandle</u></a>	
Structure to use for a data container instance	15
<a href="#"><u>_VsxDevice</u></a>	
Declare device informations	15
<a href="#"><u>_VsxDeviceList</u></a>	
List of devices	17
<a href="#"><u>_VsxDisparityDescriptor2</u></a>	
Disparity descriptor to calculate 3D data from disparity map	18
<a href="#"><u>_VsxImage</u></a>	
Declaration of image data	19
<a href="#"><u>_VsxLineCoordinate</u></a>	
Declare coordinate point of line	21

<a href="#">_VsxLineData</a>	
Declare a line package	22
<a href="#">_VsxOlr2CaptureInformation</a>	
Contains information about image capture	23
<a href="#">_VsxOlr2ModbusData</a>	
Contains information about image capture	26
<a href="#">_VsxParameter</a>	
Declares parameter	27
<a href="#">_VsxParameterEnumItem</a>	
Single item of a parameter enum	28
<a href="#">_VsxParameterList</a>	
List of parameter	29
<a href="#">_VsxStatusItem</a>	
Declaration of status item	29
<a href="#">_VsxStatusItemList</a>	
List of status items	31
<a href="#">_VsxSystemHandle</a>	
Structure to use for sensor instance	32
<a href="#">_VsxTagList</a>	
List of all possible tags inside a dynamic container	32
<a href="#">_VsxTransformation</a>	
Transformation containg translation and quaternion	33

## 7 File Index

### 7.1 File List

Here is a list of all files with brief descriptions:

<a href="#">dnne.h</a>	34
<a href="#">PF.VsxProtocolDriver.WrapperNE.h</a>	38

## 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 _VsxDDeviceList::length
```

#### devices

```
const VsxDDevice* _VsxDDeviceList::devices
```

## 8.5 \_VsxDDisparityDescriptor2 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 _VsxDDisparityDescriptor2::focalLength
```

#### principalPointU

```
double _VsxDDisparityDescriptor2::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` `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`

### 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](#)
- [VsxParameterValue](#) 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.](#)

```
00001 // Copyright 2020 Aaron R Robinson
00002 //
00003 // Permission is hereby granted, free of charge, to any person obtaining a copy
00004 // of this software and associated documentation files (the "Software"), to deal
00005 // in the Software without restriction, including without limitation the rights
00006 // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00007 // copies of the Software, and to permit persons to whom the Software is furnished
00008 // to do so, subject to the following conditions:
00009 //
00010 // The above copyright notice and this permission notice shall be included in all
00011 // copies or substantial portions of the Software.
00012 //
00013 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
00014 // INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
00015 // PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
00016 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
00017 // OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
00018 // SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00019
00020 #ifndef __SRC_PLATFORM_DNNE_H__
00021 #define __SRC_PLATFORM_DNNE_H__
00022
00023 // Define our platform
00024 #ifdef _WIN32
00025     #define DNNE_WINDOWS
00026 #elif defined(__APPLE__)
00027     #define DNNE_OSX
00028 #elif defined(__FreeBSD__)
00029     #define DNNE_FREEBSD
00030 #else
00031     #define DNNE_LINUX
00032 #endif
00033
00034 // Define some platform macros
00035 #ifdef DNNE_WINDOWS
00036     #define DNNE_API __declspec(dllexport)
00037     #define DNNE_CALLTYPE __stdcall
00038     #define DNNE_CALLTYPE_CDECL __cdecl
00039     #define DNNE_CALLTYPE_STDCALL __stdcall
00040     #define DNNE_CALLTYPE_THISCALL __thiscall
00041     #define DNNE_CALLTYPE_FASTCALL __fastcall
00042     #define _DNNE_STR(s1) L ## s1
00043     #define DNNE_STR(s) _DNNE_STR(s)
00044     #define DNNE_WCHAR wchar_t
00045 #else
```



```

00046     #define DNNE_API __attribute__((__visibility__("default")))
00047     #define DNNE_CALLTYPE
00048     #define DNNE_CALLTYPE_CDECL
00049     #ifdef __i386__
00050         #define DNNE_CALLTYPE_STDCALL __attribute__((stdcall))
00051         #define DNNE_CALLTYPE_THISCALL __attribute__((thiscall))
00052         #define DNNE_CALLTYPE_FASTCALL __attribute__((fastcall))
00053     #else
00054         #define DNNE_CALLTYPE_STDCALL
00055         #define DNNE_CALLTYPE_THISCALL
00056         #define DNNE_CALLTYPE_FASTCALL
00057     #endif
00058     #define DNNE_STR(s) s
00059     #define DNNE_WCHAR uint16_t
00060 #endif
00061
00062 // Override the DNNE_API macro.
00063 // This is typically used to dictate the export semantics of functions.
00064 #ifdef DNNE_API_OVERRIDE
00065     #undef DNNE_API
00066     #define DNNE_API DNNE_API_OVERRIDE
00067 #endif
00068
00069 //
00070 // Public exports
00071 //
00072 #define DNNE_SUCCESS 0
00073
00074 enum failure_type
00075 {
00076     failure_load_runtime = 1,
00077     failure_load_export,
00078 };
00079 typedef void (DNNE_CALLTYPE* failure_fn)(enum failure_type type, int error_code);
00080
00081 #ifdef __cplusplus
00082     #define DNNE_EXTERN_C extern "C"
00083     DNNE_EXTERN_C
00084     {
00085     #else
00086     #define DNNE_EXTERN_C
00087 #endif
00088
00089 // Provide a callback for any catastrophic failures.
00090 // The provided callback will be the last call prior to a rude-abort of the process.
00091 // See dnne_abort().
00092 DNNE_API void DNNE_CALLTYPE set_failure_callback(failure_fn cb);
00093
00094 // Preload the runtime.
00095 // The runtime is lazily loaded whenever the first export is called. This function
00096 // preloads the runtime independent of calling any export and avoids the startup
00097 // cost associated with calling an export for the first time.
00098 // If the runtime fails to load, dnne_abort() will be called.
00099 DNNE_API void DNNE_CALLTYPE preload_runtime(void);
00100
00101 // Attempt to preload the runtime.
00102 // The runtime is lazily loaded whenever the first export is called. This function
00103 // preloads the runtime independent of calling any export and avoids the startup
00104 // cost associated with calling an export for the first time.
00105 // If the runtime fails to load, an error code will be returned.
00106 DNNE_API int DNNE_CALLTYPE try_preload_runtime(void);
00107
00108 // Users can override DNNE's rude-abort behavior by providing their own dnne_abort() at link time.
00109 // It is expected this function will not return. If it does return, the behavior is undefined.
00110 extern DNNE_API void dnne_abort(enum failure_type type, int error_code);
00111
00112 #ifdef __cplusplus
00113     }
00114 #endif
00115
00116 #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 [\\_VsxDatContainerHandle](#)  
*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 definition for session message received.*
- typedef struct [\\_VsxDataContainerHandle](#) VsxDatContainerHandle  
*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 }
- 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\\_NOT\\_AVAILABLE](#) = -0x8003 ,  
[VSX\\_STATUS\\_ERROR\\_VSX\\_SYSTEM\\_HANDLE\\_NOT\\_AVAILABLE](#) = -0x8004 , [VSX\\_STATUS\\_ERROR\\_MISSING\\_IP\\_ADDRESS](#) = -0x8005 , [VSX\\_STATUS\\_ERROR\\_MISSING\\_SERIALPORT\\_DECLARATION](#) = -0x8006 , [VSX\\_STATUS\\_ERROR\\_VSX\\_SYSTEM\\_HANDLE\\_NOT\\_AVAILABLE](#) = -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\\_NOT\\_AVAILABLE](#) = -0x8019 ,  
[VSX\\_STATUS\\_ERROR\\_IMAGE\\_TAG\\_ZERO](#) = -0x801B ,  
[VSX\\_STATUS\\_ERROR\\_UNABLE\\_TO\\_FIND\\_VSX\\_DATA\\_CONTAINER](#) = -0x801C , [VSX\\_STATUS\\_ERROR\\_UNABLE\\_TO\\_FIND\\_IMAGE\\_TAG\\_TO\\_DATA\\_FORMAT](#) = -0x801D ,  
[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_Y_ID_TO_DATA_FORMAT = -0x8025 , VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_TO_DATA_FORMAT = -0x8026 , VSX_STATUS_ERROR_LOG_POINTER_ZERO = -0x8027 , VSX_STATUS_ERROR_LOG_NOT_ZERO = -0x8028 , VSX_STATUS_ERROR_RESULT_NOT_ZERO = -0x8029 , VSX_STATUS_ERROR_RESULT_POINTER_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_CONFIGURATION = -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_FOUND = -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_NOT_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_NOT_ZERO = -0x804B , VSX_STATUS_ERROR_VSX_STATUS_ITEM_NOT_ZERO = -0x804C , VSX_STATUS_ERROR_ERROR_TEXT_POINTER_ZERO = -0x804D , VSX_STATUS_ERROR_ERROR_TEXT_NOT_ZERO = -0x804E , VSX_STATUS_ERROR_ON_DISCONNECT = -0x804F , VSX_STATUS_ERROR_MAC_ADDRESS_ZERO = -0x8050 , VSX_STATUS_ERROR_VSX_CACHED_CONTAINER_NOT_FOUND = -0x8051 , VSX_STATUS_ERROR_VSX_PARAMETER_LIST_NOT_ZERO = -0x8052 , VSX_STATUS_ERROR_VSX_PARAMETER_LIST_POINTER_ZERO = -0x8053 , VSX_STATUS_ERROR_VSX_PARAMETER_ZERO = -0x8054 , VSX_STATUS_ERROR_VSX_LINE_DATA_POINTER_ZERO = -0x8055 , VSX_STATUS_ERROR_VSX_LINE_DATA_TAG_ZERO = -0x8056 , VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_TAG_TO_DATA_FORMAT = -0x8057 , VSX_STATUS_ERROR_VSX_LINE_DATA_ZERO = -0x8058 , VSX_STATUS_ERROR_MISSING_LOGIN_PASSWORD = -0x8059 , VSX_STATUS_ERROR_MISSING_LOGIN_USERNAME = -0x805A , VSX_STATUS_ERROR_ON_SESSION_MESSAGE_RECEIVED = -0x805B , VSX_STATUS_ERROR_VSX_PARAMETER_IN_POINTER_ZERO = -0x805C , VSX_STATUS_ERROR_VSX_PARAMETER_POINTER_ZERO = -0x805D , VSX_STATUS_ERROR_VSX_VALUE_POINTER_ZERO = -0x805E , VSX_STATUS_ERROR_DATA_POINTER_ZERO = -0x805F , VSX_STATUS_ERROR_UNABLE_TO_FIND_MESSAGE_IN_DATA_CONTAINER = -0x8060 , VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_NOT_ZERO = -0x8061 , VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_ZERO = -0x8062 , VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG_IN_DATA_CONTAINER = -0x8063 , VSX_STATUS_ERROR_INCORRECT_MESSAGE_FROM_TAG = -0x8064 }

```

*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 }

*Definition 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 disconnect 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_TYPE_UINT`  
`= 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 VsxCStatusCode DNNE_CALLTYPE vsx_ReleaseString (const char **p↵  
String)`

*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)`

*Init 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 \(VsxSystemHandle \\*vsx, VsxDDataContainerHandle \\*\\*pDch, int32\\_t timeout\\_ms\)](#)  
*Gets the oldest saved item and removes it internally.*
- [DNNE\\_EXTERN\\_C DNNE\\_API VsxStatusCode DNNE\\_CALLTYPE vsx\\_GetCachedContainer \(VsxSystemHandle \\*vsx, VsxDDataContainerHandle \\*\\*pDch, int32\\_t position\)](#)



*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 **pImage)`

*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 **pLineData)`

*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↔<br>_VSX_SYSTEM   |                                                                             |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE↔<br>_NOT_ZERO      |                                                                             |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE↔<br>_ZERO          |                                                                             |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE↔<br>_NOT_AVAILABLE |                                                                             |
| VSX_STATUS_ERROR_MISSING_IP_ADDRESS↔<br>_DECLARATION  |                                                                             |
| VSX_STATUS_ERROR_MISSING_SERIALPORT_↔<br>DECLARATION  |                                                                             |
| VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE↔<br>_POINTER_ZERO  |                                                                             |
| VSX_STATUS_ERROR_CONFIGURATION_ID_↔<br>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_↔<br>POINTER_ZERO       |                                                                             |
| VSX_STATUS_ERROR_OUTPUT_VALUE_NOT_↔<br>ZERO           |                                                                             |
| VSX_STATUS_ERROR_VALUE_POINTER_ZERO                   |                                                                             |

## Enumerator

|                                                                   |  |
|-------------------------------------------------------------------|--|
| VSX_STATUS_ERROR_VALUE_NOT_ZERO                                   |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX↔<br>_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↔<br>_HANDLE_POINTER_ZERO      |  |
| VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE↔<br>_VSX_DATA_CONTAINER       |  |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↔<br>_HANDLE_NOT_ZERO          |  |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↔<br>_HANDLE_ZERO              |  |
| VSX_STATUS_ERROR_VSX_DATA_CONTAINER↔<br>_HANDLE_NOT_AVAILABLE     |  |
| VSX_STATUS_ERROR_IMAGE_TAG_ZERO                                   |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX↔<br>_DATA_CONTAINER           |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>IMAGE_ID_IN_DATA_CONTAINER   |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>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_↔<br>POINT_X_ID_IN_DATA_CONTAINER |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>POINT_Y_ID_IN_DATA_CONTAINER |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>POINT_Z_ID_IN_DATA_CONTAINER |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>POINT_X_ID_TO_DATA_FORMAT    |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>POINT_Y_ID_TO_DATA_FORMAT    |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>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_↔<br>RESULT_ID_IN_DATA_CONTAINER  |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>RESULT_ID_TO_DATA_FORMAT     |  |
| VSX_STATUS_ERROR_VERSION_POINTER_ZERO                             |  |
| VSX_STATUS_ERROR_VERSION_NOT_ZERO                                 |  |
| VSX_STATUS_ERROR_VSX_IMAGE_POINTER_↔<br>ZERO                      |  |

## Enumerator

|                                                            |  |
|------------------------------------------------------------|--|
| VSX_STATUS_ERROR_VSX_IMAGE_NOT_ZERO                        |  |
| VSX_STATUS_ERROR_UNDEFINED_STRATEGY↔<br>_VALUE             |  |
| VSX_STATUS_ERROR_UNDEFINED_↔<br>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↔<br>_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↔<br>_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_↔<br>POINTER_ZERO         |  |
| VSX_STATUS_ERROR_VSX_DEVICE_LIST_ZERO                      |  |
| VSX_STATUS_ERROR_VSX_TAG_LIST_ZERO                         |  |
| VSX_STATUS_ERROR_VSX_TAG_LIST_↔<br>POINTER_ZERO            |  |
| VSX_STATUS_ERROR_VSX_TAG_LIST_NOT_↔<br>ZERO                |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_LIST↔<br>_POINTER_ZERO      |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_LIST↔<br>_ZERO              |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_NOT↔<br>_ZERO               |  |
| VSX_STATUS_ERROR_VSX_STATUS_ITEM_↔<br>LIST_POINTER_ZERO    |  |
| VSX_STATUS_ERROR_VSX_STATUS_ITEM_↔<br>LIST_ZERO            |  |
| VSX_STATUS_ERROR_VSX_STATUS_ITEM_↔<br>NOT_ZERO             |  |
| VSX_STATUS_ERROR_ERROR_TEXT_↔<br>POINTER_ZERO              |  |
| VSX_STATUS_ERROR_ERROR_TEXT_NOT_ZERO                       |  |
| VSX_STATUS_ERROR_ON_DISCONNECT_↔<br>CALLBACK_ZERO          |  |
| VSX_STATUS_ERROR_MAC_ADDRESS_ZERO                          |  |
| VSX_STATUS_ERROR_VSX_CACHED_↔<br>CONTAINER_NOT_FOUND       |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_LIST↔<br>_NOT_ZERO          |  |

## Enumerator

|                                                                 |  |
|-----------------------------------------------------------------|--|
| VSX_STATUS_ERROR_VSX_PARAMETER_↔<br>POINTER_ZERO                |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_ZERO                             |  |
| VSX_STATUS_ERROR_VSX_LINE_DATA_↔<br>POINTER_ZERO                |  |
| VSX_STATUS_ERROR_LINE_DATA_TAG_ZERO                             |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_↔<br>_ID_IN_DATA_CONTAINER |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_↔<br>_TAG_TO_DATA_FORMAT   |  |
| VSX_STATUS_ERROR_VSX_LINE_NOT_ZERO                              |  |
| VSX_STATUS_ERROR_VSX_LINE_DATA_ZERO                             |  |
| VSX_STATUS_ERROR_MISSING_LOGIN_↔<br>PASSWORD                    |  |
| VSX_STATUS_ERROR_MISSING_LOGIN_↔<br>USERNAME                    |  |
| VSX_STATUS_ERROR_ON_SESSION_↔<br>MESSAGE_RECEIVED_CALLBACK_ZERO |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_IN_↔<br>POINTER_ZERO             |  |
| VSX_STATUS_ERROR_VSX_PARAMETER_OUT_↔<br>_POINTER_ZERO           |  |
| VSX_STATUS_ERROR_VSX_VALUE_POINTER_↔<br>ZERO                    |  |
| VSX_STATUS_ERROR_DATA_POINTER_ZERO                              |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_↔<br>MESSAGE_IN_DATA_CONTAINER  |  |
| VSX_STATUS_ERROR_DATA_POINTER_↔<br>CONTENTS_NOT_ZERO            |  |
| VSX_STATUS_ERROR_DATA_POINTER_↔<br>CONTENTS_ZERO                |  |
| VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG                             |  |
| VSX_STATUS_ERROR_INCORRECT_MESSAGE_↔<br>_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 disconnect 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`

Definition 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

|                |                             |
|----------------|-----------------------------|
| <i>pString</i> | 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**

|                |                             |
|----------------|-----------------------------|
| <i>version</i> | 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**

|                   |                             |
|-------------------|-----------------------------|
| <i>error_code</i> | Input error code            |
| <i>error_text</i> | 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**

|                   |                                                                                                                    |
|-------------------|--------------------------------------------------------------------------------------------------------------------|
| <i>pVsx</i>       | Reference to an empty vsx system handle                                                                            |
| <i>ipAddress</i>  | e.g. 192.168.2.4                                                                                                   |
| <i>pluginName</i> | 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 )
```

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

**Parameters**

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

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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**

|                  |                    |
|------------------|--------------------|
| <i>vsx</i>       | handle to sensor   |
| <i>ipAddress</i> | 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**

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

|                       |                          |
|-----------------------|--------------------------|
| <i>vsx</i>            | handle to sensor         |
| <i>serialPort</i>     | The new serial port.     |
| <i>baudrate</i>       | The new baudrate.        |
| <i>connectionType</i> | 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**

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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**

|                   |                                      |
|-------------------|--------------------------------------|
| <i>vsx</i>        | handle to sensor                     |
| <i>timeout_ms</i> | 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**

|                 |                    |
|-----------------|--------------------|
| <i>vsx</i>      | handle to sensor   |
| <i>username</i> | username for login |
| <i>password</i> | 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**

|                   |                                      |
|-------------------|--------------------------------------|
| <i>vsx</i>        | handle to sensor                     |
| <i>username</i>   | username for login                   |
| <i>password</i>   | password for login                   |
| <i>timeout_ms</i> | 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**

|                 |                    |
|-----------------|--------------------|
| <i>vsx</i>      | handle to sensor   |
| <i>username</i> | username for login |
| <i>password</i> | 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

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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

|                              |                                          |
|------------------------------|------------------------------------------|
| <i>vsx</i>                   | handle to sensor                         |
| <i>authorizationUsername</i> | username for authorization account       |
| <i>authorizationPassword</i> | password for authorization account       |
| <i>username</i>              | username for account to set new password |
| <i>password</i>              | 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

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>vsx</i>    | Handle to sensor                                        |
| <i>result</i> | Pointer to result value (1: connected, 0: disconnected) |

## 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**

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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**

|             |                                       |
|-------------|---------------------------------------|
| <i>vsx</i>  | Handle to vsx sensor                  |
| <i>fptr</i> | 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**

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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**

|             |                                       |
|-------------|---------------------------------------|
| <i>vsx</i>  | Handle to vsx sensor                  |
| <i>fptr</i> | 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**

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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**

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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**

|                    |                                       |
|--------------------|---------------------------------------|
| <i>vsx</i>         | Handle to sensor                      |
| <i>command</i>     | The test system command.              |
| <i>inputValue</i>  | Optional input value                  |
| <i>outputValue</i> | Return output string of function call |
| <i>status</i>      | 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**

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

|                   |                  |
|-------------------|------------------|
| <i>vsx</i>        | Handle to sensor |
| <i>timeout_ms</i> | 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

|                   |                  |
|-------------------|------------------|
| <i>vsx</i>        | Handle to sensor |
| <i>timeout_ms</i> | 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

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>vsx</i>      | Handle to sensor                        |
| <i>fileName</i> | 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**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>vsx</i>      | Handle to sensor                            |
| <i>fileName</i> | 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**

|                   |                  |
|-------------------|------------------|
| <i>vsx</i>        | Handle to sensor |
| <i>xmlCommand</i> | 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**

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

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

|                   |                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------|
| <i>vsx</i>        | Handle to sensor                                                                                |
| <i>bufferSize</i> | The maximum number of items which will be internally saved, if less than 0, number is infinity. |
| <i>strategy</i>   | 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

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>vsx</i>        | Handle to sensor                               |
| <i>pDch</i>       | New dynamic container handle                   |
| <i>timeout_ms</i> | 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

|                 |                                        |
|-----------------|----------------------------------------|
| <i>vsx</i>      | Handle to sensor                       |
| <i>pDch</i>     | Handle to new generated data container |
| <i>position</i> | 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

|            |                          |
|------------|--------------------------|
| <i>dch</i> | Handle to data container |
|------------|--------------------------|

#### Returns

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

**vsx\_SaveData()**

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

Saves a VsxMessage to the given filename.

**Parameters**

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>dch</i>      | Handle to dynamic container                                                    |
| <i>tag</i>      | Specify which tag from container should be saved ("*" save complete container) |
| <i>fileName</i> | 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 (
    VsxDatContainerHandle * 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**

|                                           |                                           |
|-------------------------------------------|-------------------------------------------|
| <i>dch</i>                                | Handle to dynamic container               |
| <i>point_x</i> <sub>↔</sub><br><i>_Id</i> | The x image tag name                      |
| <i>point_y</i> <sub>↔</sub><br><i>_Id</i> | The y image tag name                      |
| <i>point_z</i> <sub>↔</sub><br><i>_Id</i> | The z image tag name                      |
| <i>fileName</i>                           | 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 (
    VsxDatContainerHandle * dch,
```

```
const char * tag,  
VsxImage ** imageData )
```

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

#### Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>dch</i>       | Handle to dynamic container |
| <i>tag</i>       | Tag name of image data      |
| <i>imageData</i> | 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

|               |                        |
|---------------|------------------------|
| <i>pImage</i> | Handle to image object |
|---------------|------------------------|

#### Returns

### **vsx\_GetLine()**

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

Get line data from a dynamic container.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>dch</i>  | Handle to dynamic container |
| <i>tag</i>  | Tag name of line data       |
| <i>data</i> | 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**

|                  |                            |
|------------------|----------------------------|
| <i>pLineData</i> | Handle to line data object |
|------------------|----------------------------|

**Returns****vsx\_GetDisparityDescriptor2()**

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2 (
    VsxDynamicContainerHandle * dch,
    const char * tag,
    VsxDynamicDescriptor2 ** data )
```

Get disparity descriptor from a dynamic container.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>dch</i>  | Handle to dynamic container |
| <i>tag</i>  | Tag name of data            |
| <i>data</i> | 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 (
    VsxDynamicDescriptor2 ** pData )
```

Release / free DisparityDescriptor2 object.

**Parameters**

|              |                       |
|--------------|-----------------------|
| <i>pData</i> | Handle to data object |
|--------------|-----------------------|



## Returns

### vsx\_GetTransformation()

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

Get transformation from a dynamic container.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>dch</i>  | Handle to dynamic container |
| <i>tag</i>  | Tag name of data            |
| <i>data</i> | 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

|              |                       |
|--------------|-----------------------|
| <i>pData</i> | Handle to data object |
|--------------|-----------------------|

## Returns

### vsx\_GetCaptureInformation()

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

Get capture information from a dynamic container.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>dch</i>  | Handle to dynamic container |
| <i>tag</i>  | Tag name of data            |
| <i>data</i> | 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

|              |                       |
|--------------|-----------------------|
| <i>pData</i> | Handle to data object |
|--------------|-----------------------|

## Returns

**vsx\_GetOlr2CaptureInformation()**

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

Get olr2 capture information from a dynamic container.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>dch</i>  | Handle to dynamic container |
| <i>tag</i>  | Tag name of data            |
| <i>data</i> | 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

|              |                       |
|--------------|-----------------------|
| <i>pData</i> | Handle to data object |
|--------------|-----------------------|

#### Returns

### **vsx\_GetOlr2ModbusData()**

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

Get modbus data for olr2 sensor from a dynamic container.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>dch</i>  | Handle to dynamic container |
| <i>tag</i>  | Tag name of data            |
| <i>data</i> | 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

|              |                       |
|--------------|-----------------------|
| <i>pData</i> | 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

|                |                               |
|----------------|-------------------------------|
| <i>dch</i>     | Handle to dynamic container   |
| <i>tagList</i> | 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

|                 |                           |
|-----------------|---------------------------|
| <i>pTagList</i> | 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

|               |                         |
|---------------|-------------------------|
| <i>vsx</i>    | Handle to sensor        |
| <i>result</i> | 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**

|               |                         |
|---------------|-------------------------|
| <i>vsx</i>    | Handle to sensor        |
| <i>result</i> | 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**

|               |                         |
|---------------|-------------------------|
| <i>vsx</i>    | Handle to sensor        |
| <i>result</i> | 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**

|                   |                                  |
|-------------------|----------------------------------|
| <i>vsx</i>        | Handle to sensor                 |
| <i>deviceData</i> | 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**

|                |                                  |
|----------------|----------------------------------|
| <i>pDevice</i> | Handle to existing device object |
|----------------|----------------------------------|

**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**

|                       |                                  |
|-----------------------|----------------------------------|
| <i>deviceListData</i> | 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**

|                    |                                       |
|--------------------|---------------------------------------|
| <i>pDeviceList</i> | 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

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

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>vsx</i>        | Handle to sensor                               |
| <i>log</i>        | New handle to string list                      |
| <i>timeout_ms</i> | 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**

|               |                         |
|---------------|-------------------------|
| <i>vsx</i>    | Handle to sensor        |
| <i>result</i> | Pointer to result value |

**Returns**

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

**vsx\_GetMissingLogMessagesCounter()**

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

Gets the missing log messages counter for log message grabbing.

**Parameters**

|               |                         |
|---------------|-------------------------|
| <i>vsx</i>    | Handle to sensor        |
| <i>result</i> | Pointer to result value |

**Returns**

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

**vsx\_SetSingleParameterValue()**

```
DNNE_EXTERN_C DNNE_API VsxCStatusCode 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**

|                             |                                                                         |
|-----------------------------|-------------------------------------------------------------------------|
| <i>vsx</i>                  | Handle to sensor                                                        |
| <i>settingsVersion</i>      | The settings version of the parameter which should be set.              |
| <i>configurationId</i>      | The config id of the parameter which should be set.                     |
| <i>configurationVersion</i> | The config version of the parameter which should be set.                |
| <i>parameterId</i>          | The id of the parameter which should be set.                            |
| <i>value</i>                | 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**

|                             |                                                            |
|-----------------------------|------------------------------------------------------------|
| <i>vsx</i>                  | Handle to sensor                                           |
| <i>settingsVersion</i>      | The settings version of the parameter which should be set. |
| <i>configurationId</i>      | The config id of the parameter which should be set.        |
| <i>configurationVersion</i> | The config version of the parameter which should be set.   |
| <i>parameterId</i>          | The id of the parameter which should be set.               |
| <i>value</i>                | 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**

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

**Returns**

Returns `VSX_STATUS_SUCCESS(0)` on success

**vsx\_GetSingleParameterValue()**

```
DNNE_EXTERN_C DNNE_API VsxCStatusCode 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**

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

**Returns**

Returns `VSX_STATUS_SUCCESS(0)` on success

**vsx\_GetSingleParameterValueDouble()**

```
DNNE_EXTERN_C DNNE_API VsxCStatusCode 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**

|                             |                                                               |
|-----------------------------|---------------------------------------------------------------|
| <i>vsx</i>                  | Handle to sensor                                              |
| <i>settingsVersion</i>      | The settings version of the parameter its value is asked for. |
| <i>configurationId</i>      | The config id of the parameter its value is asked for.        |
| <i>configurationVersion</i> | The config version of the parameter its value is asked for.   |
| <i>parameterId</i>          | The id of the parameter its value is asked for.               |
| <i>value</i>                | Returns value in double representation                        |

**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**

|                             |                                                               |
|-----------------------------|---------------------------------------------------------------|
| <i>vsx</i>                  | Handle to sensor                                              |
| <i>settingsVersion</i>      | The settings version of the parameter its value is asked for. |
| <i>configurationId</i>      | The config id of the parameter its value is asked for.        |
| <i>configurationVersion</i> | The config version of the parameter its value is asked for.   |
| <i>parameterId</i>          | The id of the parameter its value is asked for.               |
| <i>value</i>                | Returns value in int32 representation                         |

**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**

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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

|            |                  |
|------------|------------------|
| <i>vsx</i> | 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

|                 |                              |
|-----------------|------------------------------|
| <i>vsx</i>      | Handle to sensor             |
| <i>fileName</i> | 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**

|                 |                                |
|-----------------|--------------------------------|
| <i>vsx</i>      | Handle to sensor               |
| <i>fileName</i> | 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**

|                          |                                          |
|--------------------------|------------------------------------------|
| <i>vsx</i>               | Handle to sensor                         |
| <i>parameterListData</i> | 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**

|                          |                                               |
|--------------------------|-----------------------------------------------|
| <i>vsx</i>               | Handle to sensor                              |
| <i>parameterListData</i> | 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

|                  |                                             |
|------------------|---------------------------------------------|
| <i>vsx</i>       | Handle to sensor                            |
| <i>parameter</i> | The parameter the value should be set from. |
| <i>value</i>     | 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

|                  |                                             |
|------------------|---------------------------------------------|
| <i>vsx</i>       | Handle to sensor                            |
| <i>parameter</i> | The parameter the value should be set from. |
| <i>value</i>     | 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

|                  |                                             |
|------------------|---------------------------------------------|
| <i>vsx</i>       | Handle to sensor                            |
| <i>parameter</i> | The parameter the value should be set from. |
| <i>value</i>     | 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**

|                     |                                                                             |
|---------------------|-----------------------------------------------------------------------------|
| <i>vsx</i>          | Handle to sensor                                                            |
| <i>parameterIn</i>  | The parameter its value is asked for                                        |
| <i>parameterOut</i> | 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**

|                   |                     |
|-------------------|---------------------|
| <i>pParameter</i> | 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**

|                       |                          |
|-----------------------|--------------------------|
| <i>pParameterList</i> | Handle to parameter list |
|-----------------------|--------------------------|

**Returns**

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

**vsx\_GetResultXml()**

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

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

**Parameters**

|                 |                                       |
|-----------------|---------------------------------------|
| <i>dch</i>      | Handle to data container              |
| <i>resultId</i> | Name of result                        |
| <i>result</i>   | 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 (
    VsxDatContainerHandle * dch,
    const char * resultId,
    const char * xPath,
    const char ** result )
```

Return certain value from a result inside data container.

**Parameters**

|                 |                          |
|-----------------|--------------------------|
| <i>dch</i>      | Handle to data container |
| <i>resultId</i> | Name of result           |
| <i>xPath</i>    | xPath definition         |
| <i>result</i>   | Return result as string  |

**Returns**

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

**vsx\_GetResultElementInt32()**

```
DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32 (
    VsxDatContainerHandle * dch,
```

```
const char * resultId,  
const char * xPath,  
int32_t * result )
```

Return certain value from a result inside data container.

#### Parameters

|                   |                          |
|-------------------|--------------------------|
| <i>dch</i>        | Handle to data container |
| <i>resultId</i> ↔ | Name of result           |
| <i>xPath</i>      | xPath defintion          |
| <i>result</i>     | Return result as int32   |

#### Returns

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

### vsx\_GetResultElementInt64()

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

Return certain value from a result inside data container.

#### Parameters

|                   |                          |
|-------------------|--------------------------|
| <i>dch</i>        | Handle to data container |
| <i>resultId</i> ↔ | Name of result           |
| <i>xPath</i>      | xPath defintion          |
| <i>result</i>     | Return result as int64   |

#### Returns

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

### vsx\_GetResultElementDouble()

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

Return certain value from a result inside data container.

## Parameters

|                              |                          |
|------------------------------|--------------------------|
| <i>dch</i>                   | Handle to data container |
| <i>result</i> ↔<br><i>Id</i> | Name of result           |
| <i>xPath</i>                 | xPath definition         |
| <i>result</i>                | 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

|                           |                                  |
|---------------------------|----------------------------------|
| <i>vsx</i>                | handle to sensor                 |
| <i>statusItemListData</i> | 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

|                        |                              |
|------------------------|------------------------------|
| <i>pStatusItemList</i> | 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

|             |                                       |
|-------------|---------------------------------------|
| <i>vsx</i>  | Handle to vsx sensor                  |
| <i>fptr</i> | Function pointer to callback function |

#### Returns

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

### **vsx\_DeregisterOnDeviceStatusReceived()**

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

Function to deregister already existing callback function.

#### Parameters

|            |                  |
|------------|------------------|
| <i>vsx</i> | Handle to sensor |
|------------|------------------|

#### Returns

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

### **vsx\_SubscribeToDeviceStatusData()**

```
DNNE_EXTERN_C DNNE_API VsxCStatusCode 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

|            |                  |
|------------|------------------|
| <i>vsx</i> | Handle to sensor |
|------------|------------------|

#### Returns

Returns [VSX\\_STATUS\\_SUCCESS\(0\)](#) on success

### **vsx\_UnsubscribeToDeviceStatusData()**

```
DNNE_EXTERN_C DNNE_API VsxCStatusCode 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.](#)

```

00001 //
00002 // Auto-generated by dnne-gen
00003 //
00004 // .NET Assembly: PF.VsxProtocolDriver.Wrapper
00005 //
00006 //
00007 //
00008 // Declare exported functions
00009 //
00010 #ifndef __DNNE_GENERATED_HEADER_PF_VSXPROTOCOLDRIVER_WRAPPER__
00011 #define __DNNE_GENERATED_HEADER_PF_VSXPROTOCOLDRIVER_WRAPPER__
00012
00013 #include <stddef.h>
00014 #include <stdint.h>
00015 #include <dnne.h>
00016
00017 //
00018 // Additional code provided by user
00019 //
00020
00024 #ifdef _CVI_
00025 typedef struct _big64
00026 {
00027     unsigned int x;
00028     unsigned int y;
00029 }
00030 big64;
00031 #define LOCAL_INT64_T big64
00032 typedef struct _unsigned_big64
00033 {
00034     unsigned int x;
00035     unsigned int y;
00036 }
00037 unsigned_big64;
00038 #define LOCAL_UINT64_T unsigned_big64
00039 #else
00040 #define LOCAL_INT64_T int64_t
00041 #define LOCAL_UINT64_T uint64_t
00042 #endif
00043
00045 typedef enum _vsxStrategy
00046 {
00048     VSX_STRATEGY_DROP_OLDEST = 0,
00050     VSX_STRATEGY_DROP_WRITE = 1
00051 }
00052 VsxStrategy;
00053
00055 typedef enum _vsxStatusCode {
00057     VSX_STATUS_SUCCESS = 0,
00058
00059     VSX_STATUS_ERROR_DRIVER_INIT = -0x1,
00060     VSX_STATUS_ERROR_DRIVER_TIMEOUT = -0x2,
00061     VSX_STATUS_ERROR_DRIVER_SAVE_FILE = -0x3,
00062     VSX_STATUS_ERROR_DRIVER_DATA = -0x4,
00063     VSX_STATUS_ERROR_DRIVER_CONNECTION = -0x5,
00064     VSX_STATUS_ERROR_DRIVER_INVALID_DATA = -0x6,
00065     VSX_STATUS_ERROR_DRIVER_DEVICE = -0x7,
00066     VSX_STATUS_ERROR_DRIVER_LOAD_FILE = -0x8,
00067     VSX_STATUS_ERROR_SESSION = -0x9,
00068     VSX_STATUS_ERROR_STRING = -0x0A,
00069     VSX_STATUS_ERROR_VERSION = -0x0B,
00070     VSX_STATUS_ERROR_DRIVER_GENERAL = -0x1000,
00071
00072     VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_SYSTEM = -0x8001,
00073     VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_ZERO = -0x8002,
00074     VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_ZERO = -0x8003,
00075     VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_NOT_AVAILABLE = -0x8004,

```

```
00076     VSX_STATUS_ERROR_MISSING_IP_ADDRESS_DECLARATION = -0x8005,
00077     VSX_STATUS_ERROR_MISSING_SERIALPORT_DECLARATION = -0x8006,
00078     VSX_STATUS_ERROR_VSX_SYSTEM_HANDLE_POINTER_ZERO = -0x8007,
00079     VSX_STATUS_ERROR_CONFIGURATION_ID_ZERO = -0x8008,
00080     VSX_STATUS_ERROR_PARAMETER_ID_ZERO = -0x8009,
00081     VSX_STATUS_ERROR_VALUE_ZERO = -0x800A,
00082     VSX_STATUS_ERROR_COMMAND_ZERO = -0x800B,
00083     VSX_STATUS_ERROR_INPUT_VALUE_ZERO = -0x800C,
00084     VSX_STATUS_ERROR_OUTPUT_VALUE_POINTER_ZERO = -0x800D,
00085     VSX_STATUS_ERROR_OUTPUT_VALUE_NOT_ZERO = -0x800E,
00086     VSX_STATUS_ERROR_VALUE_POINTER_ZERO = -0x800F,
00087     VSX_STATUS_ERROR_VALUE_NOT_ZERO = -0x8010,
00088     VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX_SYSTEM = -0x8011,
00089     VSX_STATUS_ERROR_XML_COMMAND_ZERO = -0x8012,
00090     VSX_STATUS_ERROR_FILENAME_ZERO = -0x8013,
00091     VSX_STATUS_ERROR_STRING_POINTER_ZERO = -0x8014,
00092     VSX_STATUS_ERROR_STRING_ZERO = -0x8015,
00093     VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_POINTER_ZERO = -0x8016,
00094     VSX_STATUS_ERROR_UNABLE_TO_ALLOCATE_VSX_DATA_CONTAINER = -0x8017,
00095     VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_NOT_ZERO = -0x8018,
00096     VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_ZERO = -0x8019,
00097     VSX_STATUS_ERROR_VSX_DATA_CONTAINER_HANDLE_NOT_AVAILABLE = -0x801A,
00098     VSX_STATUS_ERROR_IMAGE_TAG_ZERO = -0x801B,
00099     VSX_STATUS_ERROR_UNABLE_TO_FIND_VSX_DATA_CONTAINER = -0x801C,
00100     VSX_STATUS_ERROR_UNABLE_TO_FIND_IMAGE_ID_IN_DATA_CONTAINER = -0x801D,
00101     VSX_STATUS_ERROR_UNABLE_TO_FIND_IMAGE_TAG_TO_DATA_FORMAT = -0x801E,
00102     VSX_STATUS_ERROR_POINT_Z_ID_ZERO = -0x801F,
00103     VSX_STATUS_ERROR_POINT_Y_ID_ZERO = -0x8020,
00104     VSX_STATUS_ERROR_POINT_X_ID_ZERO = -0x8021,
00105     VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_X_ID_IN_DATA_CONTAINER = -0x8022,
00106     VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Y_ID_IN_DATA_CONTAINER = -0x8023,
00107     VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_IN_DATA_CONTAINER = -0x8024,
00108     VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_X_ID_TO_DATA_FORMAT = -0x8025,
00109     VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Y_ID_TO_DATA_FORMAT = -0x8026,
00110     VSX_STATUS_ERROR_UNABLE_TO_FIND_POINT_Z_ID_TO_DATA_FORMAT = -0x8027,
00111     VSX_STATUS_ERROR_LOG_POINTER_ZERO = -0x8028,
00112     VSX_STATUS_ERROR_LOG_NOT_ZERO = -0x8029,
00113     VSX_STATUS_ERROR_RESULT_NOT_ZERO = -0x802A,
00114     VSX_STATUS_ERROR_RESULT_POINTER_ZERO = -0x802B,
00115     VSX_STATUS_ERROR_UNABLE_TO_FIND_RESULT_ID_IN_DATA_CONTAINER = -0x802C,
00116     VSX_STATUS_ERROR_UNABLE_TO_FIND_RESULT_ID_TO_DATA_FORMAT = -0x802D,
00117     VSX_STATUS_ERROR_VERSION_POINTER_ZERO = -0x802E,
00118     VSX_STATUS_ERROR_VERSION_NOT_ZERO = -0x802F,
00119     VSX_STATUS_ERROR_VSX_IMAGE_POINTER_ZERO = -0x8030,
00120     VSX_STATUS_ERROR_VSX_IMAGE_NOT_ZERO = -0x8031,
00121     VSX_STATUS_ERROR_UNDEFINED_STRATEGY_VALUE = -0x8032,
00122     VSX_STATUS_ERROR_UNDEFINED_CONNECTION_TYPE_VALUE = -0x8033,
00123     VSX_STATUS_ERROR_XPATH_ZERO = -0x8034,
00124     VSX_STATUS_ERROR_INVALID_DATA_FORMAT = -0x8035,
00125     VSX_STATUS_ERROR_NO_ELEMENT_FOUND = -0x8036,
00126     VSX_STATUS_ERROR_RESULT_TAG_ZERO = -0x8037,
00127     VSX_STATUS_ERROR_TAG_ZERO = -0x8038,
00128     VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG_IN_DATA_CONTAINER = -0x8039,
00129     VSX_STATUS_ERROR_IP_ADDRESS_ZERO = -0x803A,
00130     VSX_STATUS_ERROR_NETWORK_MASK_ZERO = -0x803B,
00131     VSX_STATUS_ERROR_GATEWAY_ZERO = -0x803C,
00132     VSX_STATUS_ERROR_EXCEPTION_THROWN = -0x803D,
00133     VSX_STATUS_ERROR_VSX_DEVICE_POINTER_ZERO = -0x803E,
00134     VSX_STATUS_ERROR_VSX_DEVICE_NOT_ZERO = -0x803F,
00135     VSX_STATUS_ERROR_VSX_IMAGE_ZERO = -0x8040,
00136     VSX_STATUS_ERROR_VSX_DEVICE_ZERO = -0x8041,
00137     VSX_STATUS_ERROR_VSX_DEVICE_LIST_POINTER_ZERO = -0x8042,
00138     VSX_STATUS_ERROR_VSX_DEVICE_LIST_ZERO = -0x8043,
00139     VSX_STATUS_ERROR_VSX_TAG_LIST_ZERO = -0x8044,
00140     VSX_STATUS_ERROR_VSX_TAG_LIST_POINTER_ZERO = -0x8045,
00141     VSX_STATUS_ERROR_VSX_TAG_LIST_NOT_ZERO = -0x8046,
00142     VSX_STATUS_ERROR_VSX_PARAMETER_LIST_POINTER_ZERO = -0x8047,
00143     VSX_STATUS_ERROR_VSX_PARAMETER_LIST_ZERO = -0x8048,
00144     VSX_STATUS_ERROR_VSX_PARAMETER_NOT_ZERO = -0x8049,
00145     VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_POINTER_ZERO = -0x804A,
00146     VSX_STATUS_ERROR_VSX_STATUS_ITEM_LIST_ZERO = -0x804B,
00147     VSX_STATUS_ERROR_VSX_STATUS_ITEM_NOT_ZERO = -0x804C,
00148     VSX_STATUS_ERROR_ERROR_TEXT_POINTER_ZERO = -0x804D,
00149     VSX_STATUS_ERROR_ERROR_TEXT_NOT_ZERO = -0x804E,
00150     VSX_STATUS_ERROR_ON_DISCONNECT_CALLBACK_ZERO = -0x804F,
00151     VSX_STATUS_ERROR_MAC_ADDRESS_ZERO = -0x8050,
00152     VSX_STATUS_ERROR_VSX_CACHED_CONTAINER_NOT_FOUND = -0x8051,
00153     VSX_STATUS_ERROR_VSX_PARAMETER_LIST_NOT_ZERO = -0x8052,
00154     VSX_STATUS_ERROR_VSX_PARAMETER_POINTER_ZERO = -0x8053,
00155     VSX_STATUS_ERROR_VSX_PARAMETER_ZERO = -0x8054,
00156     VSX_STATUS_ERROR_VSX_LINE_DATA_POINTER_ZERO = -0x8055,
00157     VSX_STATUS_ERROR_VSX_LINE_DATA_TAG_ZERO = -0x8056,
00158     VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_ID_IN_DATA_CONTAINER = -0x8057,
00159     VSX_STATUS_ERROR_UNABLE_TO_FIND_LINE_TAG_TO_DATA_FORMAT = -0x8058,
00160     VSX_STATUS_ERROR_VSX_LINE_NOT_ZERO = -0x8059,
00161     VSX_STATUS_ERROR_VSX_LINE_DATA_ZERO = -0x805A,
00162     VSX_STATUS_ERROR_MISSING_LOGIN_PASSWORD = -0x805B,
```

```

00163     VSX_STATUS_ERROR_MISSING_LOGIN_USERNAME = -0x0806B,
00164     VSX_STATUS_ERROR_ON_SESSION_MESSAGE_RECEIVED_CALLBACK_ZERO = -0x0806C,
00165     VSX_STATUS_ERROR_VSX_PARAMETER_IN_POINTER_ZERO = -0x0806D,
00166     VSX_STATUS_ERROR_VSX_PARAMETER_OUT_POINTER_ZERO = -0x0806E,
00167     VSX_STATUS_ERROR_VSX_VALUE_POINTER_ZERO = -0x0806F,
00168     VSX_STATUS_ERROR_DATA_POINTER_ZERO = -0x08070,
00169     VSX_STATUS_ERROR_UNABLE_TO_FIND_MESSAGE_IN_DATA_CONTAINER = -0x08071,
00170     VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_NOT_ZERO = -0x08072,
00171     VSX_STATUS_ERROR_DATA_POINTER_CONTENTS_ZERO = -0x08073,
00172     VSX_STATUS_ERROR_UNABLE_TO_FIND_TAG = -0x08074,
00173     VSX_STATUS_ERROR_INCORRECT_MESSAGE_FROM_TAG = -0x08075,
00174 } VsxStatusCode;
00175
00176 typedef enum _vsxSerialConnectionType {
00177     VSX_SERIAL_CONNECTION_TYPE_USB_SSI = 0,
00178     VSX_SERIAL_CONNECTION_TYPE_PROFIBUS = 1,
00179     VSX_SERIAL_CONNECTION_TYPE_PROFINET = 2,
00180     VSX_SERIAL_CONNECTION_TYPE_ETHERNET_IP = 3,
00181     VSX_SERIAL_CONNECTION_TYPE_RS485 = 4,
00182     VSX_SERIAL_CONNECTION_TYPE_CANOPEN = 5
00183 } VsxSerialConnectionType;
00184
00185 typedef struct _VsxSystemHandle
00186 {
00187     int handle;
00188 } VsxSystemHandle;
00189
00190 typedef enum _vsxDisconnectEvent {
00191     VSX_DISCONNECT_EVENT_REMOTE_HOST_CONNECTION_CLOSED = 0,
00192     VSX_DISCONNECT_EVENT_DISCONNECT_CALLED = 1,
00193     VSX_DISCONNECT_EVENT_CONNECTION_ERROR = 2
00194 } VsxDisconnectEvent;
00195
00196 typedef void (*vsx_OnDisconnect) (int handle, const char* ipAddress, VsxDisconnectEvent
00197 disconnectEvent, const char* description);
00198
00199 typedef enum _vsxSessionTypes {
00200     VSX_SESSION_TYPES_LOGIN_REQUIRED = 0,
00201     VSX_SESSION_TYPES_INITIAL_PASSWORD_REQUIRED = 1,
00202     VSX_SESSION_TYPES_LOGIN = 2,
00203     VSX_SESSION_TYPES_LOGIN_REPLY = 3,
00204     VSX_SESSION_TYPES_SET_PASSWORD = 4,
00205     VSX_SESSION_TYPES_SET_PASSWORD_REPLY = 5,
00206     VSX_SESSION_TYPES_TIMEOUT_ANNOUNCEMENT = 6,
00207     VSX_SESSION_TYPES_TIMEOUT = 7,
00208     VSX_SESSION_TYPES_LOGOUT = 8,
00209     VSX_SESSION_TYPES_LOGOUT_REPLY = 9,
00210     VSX_SESSION_TYPES_UNKNOWN = 10
00211 } VsxSessionTypes;
00212
00213 typedef void (*vsx_OnSessionMessageReceived) (int handle, VsxSessionTypes sessionType, int timeout);
00214
00215 typedef struct _VsxDataContainerHandle
00216 {
00217     int handle;
00218 } VsxDataContainerHandle;
00219
00220 typedef enum _vsxImageData2Format {
00221     VSX_IMAGE_DATA2_FORMAT_MONO8 = 17301505,
00222     VSX_IMAGE_DATA2_FORMAT_CONFIDENCE8 = 17301702,
00223     VSX_IMAGE_DATA2_FORMAT_MONO12 = 17825797,
00224     VSX_IMAGE_DATA2_FORMAT_MONO16 = 17825799,
00225     VSX_IMAGE_DATA2_FORMAT_COORD3D_A16 = 17825974,
00226     VSX_IMAGE_DATA2_FORMAT_COORD3D_B16 = 17825975,
00227     VSX_IMAGE_DATA2_FORMAT_COORD3D_C16 = 17825976,
00228     VSX_IMAGE_DATA2_FORMAT_COORD3D_A32F = 18874557,
00229     VSX_IMAGE_DATA2_FORMAT_COORD3D_B32F = 18874558,
00230     VSX_IMAGE_DATA2_FORMAT_COORD3D_C32F = 18874559
00231 } VsxImageData2Format;
00232
00233 typedef struct _VsxImage{
00234     void* rawdata;
00235     VsxImageData2Format format;
00236     int width;
00237     int height;
00238     int linePitch;
00239     LOCAL_INT64_T frameCounter;
00240     double coordinateScale;
00241     double coordinateOffset;
00242     double axisMin;
00243     double axisMax;
00244     double invalidDataValue;
00245 } VsxImage;

```



```

00260
00261
00263 typedef enum _vsxLineDataFormat {
00264     VSX_LINE_DATA_FORMAT_C = 0x00,
00265     VSX_LINE_DATA_FORMAT_X = 0x01,
00266     VSX_LINE_DATA_FORMAT_Y = 0x02,
00267     VSX_LINE_DATA_FORMAT_Z = 0x04,
00268     VSX_LINE_DATA_FORMAT_Q = 0x08,
00269     VSX_LINE_DATA_FORMAT_I = 0x10,
00270     VSX_LINE_DATA_FORMAT_32BIT_MODE = 0x1000,
00271 } VsxLineDataFormat;
00272
00274 typedef struct _VsxLineCoordinate{
00275     float c;
00276     float x;
00277     float y;
00278     float z;
00279     float q;
00280     float i;
00281 } VsxLineCoordinate;
00282
00284 typedef struct _VsxLineData{
00285     VsxLineCoordinate** lines;
00286     unsigned short format;
00287     unsigned short width;
00288     unsigned short countLines;
00289     unsigned short frameCounter;
00290     float minX;
00291     float maxX;
00292     float minZ;
00293     float maxZ;
00294 } VsxLineData;
00295
00297 typedef struct _VsxDisparityDescriptor2{
00298     double focalLength;
00299     double principalPointU;
00300     double principalPointV;
00301     double baseline;
00302     double offsetLeftRectifiedToDisparityU;
00303     double offsetLeftRectifiedToDisparityV;
00304 } VsxDisparityDescriptor2;
00305
00307 typedef struct _VsxTransformation{
00308     double translationTX;
00309     double translationTY;
00310     double translationTZ;
00311     double quaternionQ0;
00312     double quaternionQ1;
00313     double quaternionQ2;
00314     double quaternionQ3;
00315 } VsxTransformation;
00316
00318 typedef struct _VsxCaptureInformation{
00319     LOCAL_UINT64_T triggerCounter;
00320     LOCAL_UINT64_T parameterId;
00321     LOCAL_UINT64_T jobId;
00322     LOCAL_INT64_T rotaryEncoder;
00323     LOCAL_UINT64_T frameCounter;
00324     LOCAL_UINT64_T timestamp;
00325     unsigned int exposureTime;
00326     unsigned int gain;
00327     unsigned char illumination;
00328     unsigned char triggerSource;
00329 } VsxCaptureInformation;
00330
00332 typedef struct _VsxOlr2CaptureInformation {
00333     LOCAL_UINT64_T frameCounter;
00334     LOCAL_UINT64_T triggerCounter;
00335     double currentPosition;
00336     LOCAL_UINT64_T ioState;
00337     LOCAL_UINT64_T timestamp;
00338     unsigned int lmaExposureTime1;
00339     unsigned int lmaExposureTime2;
00340     unsigned int lmbExposureTime1;
00341     unsigned int lmbExposureTime2;
00342     unsigned short lmaRoiOffsetX;
00343     unsigned short lmaRoiLengthX;
00344     unsigned short lmaRoiOffsetZ;
00345     unsigned short lmaRoiLengthZ;
00346     unsigned short lmbRoiOffsetX;
00347     unsigned short lmbRoiLengthX;
00348     unsigned short lmbRoiOffsetZ;
00349     unsigned short lmbRoiLengthZ;
00350 }

```

```

00360     unsigned short  autoTriggerFrameRate ;
00361     unsigned char   triggerSource;
00362 } VsxOlr2CaptureInformation;
00363
00364
00366 typedef struct _VsxOlr2ModbusData{
00367     unsigned short  activationTimer;
00368     unsigned short  compareBuffer;
00369     unsigned short  targetPosition;
00370     unsigned short  robotData[13];
00371 } VsxOlr2ModbusData;
00372
00373
00375 typedef struct _VsxTagList{
00376     int  length;
00377     const char** tags;
00378 } VsxTagList;
00379
00380
00382 typedef struct _VsxDevice{
00383     const char* ipAddress;
00384     const char* networkMask;
00385     const char* gateway;
00386     const char* macAddress;
00387     const char* firmwareVersion;
00388     const char* sensorType;
00389     const char* sensorName;
00390     int  busy;
00391     int  deviceVsxVersionMajor;
00392     int  deviceVsxVersionMinor;
00393     const char* comPort;
00394     int  baudrate;
00395     const char* headAddress;
00396     int  isLoginNeeded;
00397 } VsxDevice;
00398
00400 typedef struct _VsxDeviceList{
00401     int  length;
00402     const VsxDevice* devices;
00403 } VsxDeviceList;
00404
00405
00407 typedef enum _vsxParameterValueType {
00408     VSX_PARAMETER_VALUE_TYPE_BOOL = 0,
00409     VSX_PARAMETER_VALUE_TYPE_INT = 1,
00410     VSX_PARAMETER_VALUE_TYPE_LONG = 2,
00411     VSX_PARAMETER_VALUE_TYPE_UINT = 3,
00412     VSX_PARAMETER_VALUE_TYPE_INT16 = 4,
00413     VSX_PARAMETER_VALUE_TYPE_FLOAT = 5,
00414     VSX_PARAMETER_VALUE_TYPE_DOUBLE = 6,
00415     VSX_PARAMETER_VALUE_TYPE_STRING = 7,
00416     VSX_PARAMETER_VALUE_TYPE_HEXSTRING = 8,
00417     VSX_PARAMETER_VALUE_TYPE_BASE64 = 9,
00418     VSX_PARAMETER_VALUE_TYPE_ENUM = 10,
00419     VSX_PARAMETER_VALUE_TYPE_IP = 11,
00420     VSX_PARAMETER_VALUE_TYPE_RECTANGLE = 12,
00421     VSX_PARAMETER_VALUE_TYPE_QUAD = 13,
00422     VSX_PARAMETER_VALUE_TYPE_POINT = 14,
00423     VSX_PARAMETER_VALUE_TYPE_UNKNOWN = 15
00424 } VsxParameterValueType;
00425
00427 typedef struct _VsxParameterEnumItem{
00428     const char* id;
00429     const char* name;
00430 } VsxParameterEnumItem;
00431
00433 typedef struct _VsxParameter{
00434     const char* valueString;
00435     int  valueInt;
00436     double valueDouble;
00437     VsxParameterValueType valueType;
00438     const char* name;
00439     const char* parameterId;
00440     const char* configId;
00441     int  configVersion;
00442     int  settingsVersion;
00443     int  enumItemListLength;
00444     const VsxParameterEnumItem* enumItemList;
00445 } VsxParameter;
00446
00448 typedef struct _VsxParameterList{
00449     int  length;
00450     const VsxParameter* parameters;
00451 } VsxParameterList;
00452
00453
00455 typedef enum _vsxStatusItemValueType {

```

```

00456     VSX_STATUS_ITEM_VALUE_TYPE_BOOL = 0,
00457     VSX_STATUS_ITEM_VALUE_TYPE_INT = 1,
00458     VSX_STATUS_ITEM_VALUE_TYPE_LONG = 2,
00459     VSX_STATUS_ITEM_VALUE_TYPE_UINT = 3,
00460     VSX_STATUS_ITEM_VALUE_TYPE_INT16 = 4,
00461     VSX_STATUS_ITEM_VALUE_TYPE_FLOAT = 5,
00462     VSX_STATUS_ITEM_VALUE_TYPE_DOUBLE = 6,
00463     VSX_STATUS_ITEM_VALUE_TYPE_STRING = 7,
00464     VSX_STATUS_ITEM_VALUE_TYPE_HEXSTRING = 8,
00465     VSX_STATUS_ITEM_VALUE_TYPE_BASE64 = 9,
00466     VSX_STATUS_ITEM_VALUE_TYPE_ENUM = 10,
00467     VSX_STATUS_ITEM_VALUE_TYPE_IP = 11,
00468     VSX_STATUS_ITEM_VALUE_TYPE_RECTANGLE = 12,
00469     VSX_STATUS_ITEM_VALUE_TYPE_QUAD = 13,
00470     VSX_STATUS_ITEM_VALUE_TYPE_POINT = 14,
00471     VSX_STATUS_ITEM_VALUE_TYPE_UNKNOWN = 15
00472 } VsxStatusItemValueType;
00473
00475 typedef enum _vsxDeviceStatusScope {
00476     VSX_DEVICE_STATUS_SCOPE_FULL = 0,
00477     VSX_DEVICE_STATUS_SCOPE_MULTI = 1
00478 } VsxDeviceStatusScope;
00479
00481 typedef struct _VsxStatusItem{
00482     const char* valueString;
00483     int valueInt;
00484     double valueDouble;
00485     VsxStatusItemValueType valueType;
00486     const char* name;
00487     const char* statusItemId;
00488     const char* configurationClass;
00489     int configVersion;
00490     int settingsVersion;
00491     LOCAL_UINT64_T time;
00492     LOCAL_UINT64_T sensorTime;
00493 } VsxStatusItem;
00494
00496 typedef struct _VsxStatusItemList{
00497     int length;
00498     const VsxStatusItem* statusItems;
00499 } VsxStatusItemList;
00500
00501
00503 typedef void (*vsx_OnDeviceStatusReceived) (int handle, VsxDeviceStatusScope deviceStatusScope, const
VsxStatusItemList* statusItemListData);
00504
00505
00506 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseString
00513 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseString(const char** pString);
00514
00515 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetLibraryVersion
00522 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLibraryVersion(const char** version);
00523
00524 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetErrorText
00532 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetErrorText(int32_t error_code, const char**
error_text);
00533
00534 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitTcpSensor
00543 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitTcpSensor(VsxSystemHandle** pVsx, const
char* ipAddress, const char* pluginName);
00544
00545 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitSerialSensor
00556 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);
00557
00558 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseSensor
00564 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseSensor(VsxSystemHandle** vsx);
00565
00566 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectTcpDevice
00573 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectTcpDevice(VsxSystemHandle* vsx, const
char* ipAddress);
00574
00575 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectAndLoginTcpDevice
00584 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectAndLoginTcpDevice(VsxSystemHandle*
vsx, const char* ipAddress, const char* username, const char* password);
00585
00586 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReConnectSerialDevice
00595 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReConnectSerialDevice(VsxSystemHandle* vsx,
const char* serialPort, int32_t baudrate, VsxSerialConnectionType connectionType);
00596
00597 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Connect
00603 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Connect(VsxSystemHandle* vsx);
00604
00605 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectEx
00612 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectEx(VsxSystemHandle* vsx, int32_t
timeout_ms);

```

```

00613
00614 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectAndLogin
00622 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectAndLogin(VsxSystemHandle* vsx, const
char* username, const char* password);
00623
00624 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectExAndLogin
00633 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectExAndLogin(VsxSystemHandle* vsx, const
char* username, const char* password, int32_t timeout_ms);
00634
00635 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Login
00643 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Login(VsxSystemHandle* vsx, const char*
username, const char* password);
00644
00645 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Logout
00651 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Logout(VsxSystemHandle* vsx);
00652
00653 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetPassword
00663 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetPassword(VsxSystemHandle* vsx, const char*
authorizationUsername, const char* authorizationPassword, const char* username, const char* password);
00664
00665 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetConnected
00672 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetConnected(VsxSystemHandle* vsx, int32_t*
result);
00673
00674 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Disconnect
00680 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Disconnect(VsxSystemHandle* vsx);
00681
00682 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnDisconnect
00691 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDisconnect(VsxSystemHandle* vsx,
vsx_OnDisconnect fptr);
00692
00693 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnDisconnect
00699 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDisconnect(VsxSystemHandle* vsx);
00700
00701 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnSessionMessageReceived
00710 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_RegisterOnSessionMessageReceived(VsxSystemHandle* vsx, vsx_OnSessionMessageReceived fptr);
00711
00712 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnSessionMessageReceived
00718 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_DeregisterOnSessionMessageReceived(VsxSystemHandle* vsx);
00719
00720 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendSessionKeepAlive
00726 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendSessionKeepAlive(VsxSystemHandle* vsx);
00727
00728 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystem
00738 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystem(VsxSystemHandle* vsx, const char*
command, const char* inputValue, const char** outputValue, int32_t* status);
00739
00740 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystemEx
00751 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);
00752
00753 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetWaitTimeout
00760 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetWaitTimeout(VsxSystemHandle* vsx, int32_t*
timeout_ms);
00761
00762 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetWaitTimeout
00769 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetWaitTimeout(VsxSystemHandle* vsx, int32_t
timeout_ms);
00770
00771 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.UploadData
00778 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadData(VsxSystemHandle* vsx, const char*
fileName);
00779
00780 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendFirmware
00788 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendFirmware(VsxSystemHandle* vsx, const char*
fileName);
00789
00790 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendXmlDataMessage
00798 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendXmlDataMessage(VsxSystemHandle* vsx, const
char* xmlCommand);
00799
00800 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettings
00809 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettings(VsxSystemHandle* vsx, const
char* ipAddress, const char* networkMask, const char* gateway);
00810
00811 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettingsViaUdp
00820 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettingsViaUdp(const char*
macAddress, const char* ipAddress, const char* networkMask, const char* gateway);
00821
00822 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ResetDynamicContainerGrabber
00830 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetDynamicContainerGrabber(VsxSystemHandle*
vsx, int32_t bufferSize, VsxStrategy strategy);
00831
00832 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDataContainer
00840 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDataContainer(VsxSystemHandle* vsx,

```

```

VsxDataContainerHandle** pDch, int32_t timeout_ms);
00841
00842 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCachedContainer
00850 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCachedContainer(VsxSystemHandle* vsx,
VsxDataContainerHandle** pDch, int32_t position);
00851
00852 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDataContainer
00858 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDataContainer(VsxDataContainerHandle**
dch);
00859
00860 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.SaveData
00868 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveData(VsxDataContainerHandle* dch, const
char* tag, const char* fileName);
00869
00870 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.Save3DPointCloudData
00880 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);
00881
00882 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetImage
00890 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetImage(VsxDataContainerHandle* dch, const
char* tag, VsxImage** imageData);
00891
00892 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseImage
00898 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseImage(VsxImage** pImage);
00899
00900 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetLine
00908 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLine(VsxDataContainerHandle* dch, const
char* tag, VsxLineData** data);
00909
00910 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseLine
00916 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseLine(VsxLineData** pLineData);
00917
00918 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDisparityDescriptor2
00926 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2(VsxDataContainerHandle*
dch, const char* tag, VsxDisparityDescriptor2** data);
00927
00928 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDisparityDescriptor2
00934 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseDisparityDescriptor2(VsxDisparityDescriptor2** pData);
00935
00936 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTransformation
00944 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTransformation(VsxDataContainerHandle* dch,
const char* tag, VsxTransformation** data);
00945
00946 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTransformation
00952 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTransformation(VsxTransformation**
pData);
00953
00954 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCaptureInformation
00962 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCaptureInformation(VsxDataContainerHandle*
dch, const char* tag, VsxCaptureInformation** data);
00963
00964 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseCaptureInformation
00970 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseCaptureInformation(VsxCaptureInformation** pData);
00971
00972 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2CaptureInformation
00980 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_GetOlr2CaptureInformation(VsxDataContainerHandle* dch, const char* tag,
VsxOlr2CaptureInformation** data);
00981
00982 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2CaptureInformation
00988 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseOlr2CaptureInformation(VsxOlr2CaptureInformation** pData);
00989
00990 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2ModbusData
00998 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2ModbusData(VsxDataContainerHandle* dch,
const char* tag, VsxOlr2ModbusData** data);
00999
01000 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2ModbusData
01006 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2ModbusData(VsxOlr2ModbusData**
pData);
01007
01008 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTagList
01015 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTagList(VsxDataContainerHandle* dch,
VsxTagList** tagList);
01016
01017 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTagList
01023 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTagList(VsxTagList** pTagList);
01024
01025 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetMissingContainerFramesCounter
01032 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_GetMissingContainerFramesCounter(VsxSystemHandle* vsx, int32_t* result);
01033
01034 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDynamicContainerQueueSize
01041 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDynamicContainerQueueSize(VsxSystemHandle*
vsx, int32_t* result);

```

```

01042
01043 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetNumberOfCachedContainers
01050 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetNumberOfCachedContainers(VsxSystemHandle*
    vsx, int32_t* result);
01051
01052 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetDeviceInformation
01059 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDeviceInformation(VsxSystemHandle* vsx,
    VsxDevice** deviceData);
01060
01061 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDevice
01067 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDevice(VsxDevice** pDevice);
01068
01069 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetUdpDeviceList
01075 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetUdpDeviceList(VsxDeviceList**
    deviceListData);
01076
01077 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDeviceList
01083 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDeviceList(VsxDeviceList** pDeviceList);
01084
01085 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.ResetLogMessageGrabber
01095 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetLogMessageGrabber(VsxSystemHandle* vsx,
    int32_t bufferSize, int32_t typeMask, VsxStrategy strategy);
01096
01097 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessage
01105 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessage(VsxSystemHandle* vsx, const
    char** log, int32_t timeout_ms);
01106
01107 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessageQueueSize
01114 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessageQueueSize(VsxSystemHandle* vsx,
    int32_t* result);
01115
01116 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetMissingLogMessagesCounter
01123 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingLogMessagesCounter(VsxSystemHandle*
    vsx, int32_t* result);
01124
01125 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValue
01136 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);
01137
01138 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueDouble
01149 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);
01150
01151 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueInt32
01162 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);
01163
01164 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValue
01175 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);
01176
01177 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueDouble
01188 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);
01189
01190 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueInt32
01201 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);
01202
01203 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadDefaultParameterSetOnDevice
01210 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
    vsx_LoadDefaultParameterSetOnDevice(VsxSystemHandle* vsx);
01211
01212 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadParameterSetOnDevice
01219 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadParameterSetOnDevice(VsxSystemHandle* vsx);
01220
01221 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SaveParameterSetOnDevice
01228 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveParameterSetOnDevice(VsxSystemHandle* vsx);
01229
01230 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.UploadParameterSet
01237 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterSet(VsxSystemHandle* vsx, const
    char* fileName);
01238
01239 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.DownloadParameterSet
01246 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DownloadParameterSet(VsxSystemHandle* vsx,
    const char* fileName);
01247
01248 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetParameterList
01256 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetParameterList(VsxSystemHandle* vsx,
    VsxParameterList** parameterListData);
01257

```

```

01258 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.UploadParameterList
01265 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterList (VsxSystemHandle* vsx,
VsxParameterList* parameterListData);
01266
01267 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterString
01275 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterString (VsxSystemHandle* vsx,
const VsxParameter* parameter, const char* value);
01276
01277 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterDouble
01285 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble (VsxSystemHandle* vsx,
const VsxParameter* parameter, double value);
01286
01287 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterInt32
01295 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterInt32 (VsxSystemHandle* vsx,
const VsxParameter* parameter, int32_t value);
01296
01297 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetSingleParameter
01305 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameter (VsxSystemHandle* vsx, const
VsxParameter* parameterIn, const VsxParameter** parameterOut);
01306
01307 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameter
01313 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameter (const VsxParameter**
pParameter);
01314
01315 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameterList
01321 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameterList (VsxParameterList**
pParameterList);
01322
01323 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultXml
01331 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultXml (VsxDataContainerHandle* dch, const
char* resultId, const char** result);
01332
01333 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementString
01342 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementString (VsxDataContainerHandle*
dch, const char* resultId, const char* xPath, const char** result);
01343
01344 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt32
01353 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32 (VsxDataContainerHandle*
dch, const char* resultId, const char* xPath, int32_t* result);
01354
01355 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt64
01364 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt64 (VsxDataContainerHandle*
dch, const char* resultId, const char* xPath, LOCAL_INT64_T* result);
01365
01366 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementDouble
01375 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementDouble (VsxDataContainerHandle*
dch, const char* resultId, const char* xPath, double* result);
01376
01377 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.GetAllDeviceStatusData
01384 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetAllDeviceStatusData (VsxSystemHandle* vsx,
VsxStatusItemList** statusItemListData);
01385
01386 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.ReleaseStatusItemList
01392 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseStatusItemList (VsxStatusItemList**
pStatusItemList);
01393
01394 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.RegisterOnDeviceStatusReceived
01403 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDeviceStatusReceived (VsxSystemHandle*
vsx, vsx_OnDeviceStatusReceived fptr);
01404
01405 // Computed from
PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.DeregisterOnDeviceStatusReceived
01411 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_DeregisterOnDeviceStatusReceived (VsxSystemHandle* vsx);
01412
01413 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.SubscribeToDeviceStatusData
01421 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SubscribeToDeviceStatusData (VsxSystemHandle*
vsx);
01422
01423 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.UnsubscribeToDeviceStatusData
01429 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UnsubscribeToDeviceStatusData (VsxSystemHandle*
vsx);
01430
01431 #endif // __DNNE_GENERATED_HEADER_PF_VSXPROTOCOLDRIVER_WRAPPER__
01432
01433 //
01434 // Define exported functions
01435 //
01436 #ifndef DNNE_COMPILE_AS_SOURCE
01437
01438 #ifndef DNNE_WINDOWS
01439     #ifndef _WCHAR_T_DEFINED
01440         typedef wchar_t char_t;
01441     #else
01442         typedef unsigned short char_t;
01443     #endif
01444 #else

```



```

01445     typedef char char_t;
01446 #endif
01447
01448 //
01449 // Forward declarations
01450 //
01451
01452 extern void* get_callable_managed_function(
01453     const char_t* dotnet_type,
01454     const char_t* dotnet_type_method,
01455     const char_t* dotnet_delegate_type);
01456
01457 extern void* get_fast_callable_managed_function(
01458     const char_t* dotnet_type,
01459     const char_t* dotnet_type_method);
01460
01461 //
01462 // String constants
01463 //
01464
01465 static const char_t* t1_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExports,
PF.VsxProtocolDriver.Wrapper");
01466 static const char_t* t2_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer,
PF.VsxProtocolDriver.Wrapper");
01467 static const char_t* t3_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsDevice,
PF.VsxProtocolDriver.Wrapper");
01468 static const char_t* t4_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsLog,
PF.VsxProtocolDriver.Wrapper");
01469 static const char_t* t5_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsParameter,
PF.VsxProtocolDriver.Wrapper");
01470 static const char_t* t6_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList,
PF.VsxProtocolDriver.Wrapper");
01471 static const char_t* t7_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsResult,
PF.VsxProtocolDriver.Wrapper");
01472 static const char_t* t8_name = DNNE_STR("PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList,
PF.VsxProtocolDriver.Wrapper");
01473
01474 //
01475 // Exports
01476 //
01477
01478 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ReleaseString
01479 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseString_ptr)(const char** pString);
01480 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseString(const char** pString)
01481 {
01482     if (vsx_ReleaseString_ptr == NULL)
01483     {
01484         const char_t* methodName = DNNE_STR("ReleaseString");
01485         vsx_ReleaseString_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (const char**
pString))get_fast_callable_managed_function(t1_name, methodName);
01486     }
01487     return vsx_ReleaseString_ptr(pString);
01488 }
01489
01490 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetLibraryVersion
01491 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLibraryVersion_ptr)(const char** version);
01492 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLibraryVersion(const char** version)
01493 {
01494     if (vsx_GetLibraryVersion_ptr == NULL)
01495     {
01496         const char_t* methodName = DNNE_STR("GetLibraryVersion");
01497         vsx_GetLibraryVersion_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (const char**
version))get_fast_callable_managed_function(t1_name, methodName);
01498     }
01499     return vsx_GetLibraryVersion_ptr(version);
01500 }
01501
01502 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetErrorText
01503 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetErrorText_ptr)(int32_t error_code, const char**
error_text);
01504 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetErrorText(int32_t error_code, const char**
error_text)
01505 {
01506     if (vsx_GetErrorText_ptr == NULL)
01507     {
01508         const char_t* methodName = DNNE_STR("GetErrorText");
01509         vsx_GetErrorText_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (int32_t error_code, const char**
error_text))get_fast_callable_managed_function(t1_name, methodName);
01510     }
01511     return vsx_GetErrorText_ptr(error_code, error_text);
01512 }
01513
01514 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.InitTcpSensor
01515 static VsxStatusCode (DNNE_CALLTYPE* vsx_InitTcpSensor_ptr)(VsxSystemHandle** pVsx, const char*
ipAddress, const char* pluginName);
01516 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_InitTcpSensor(VsxSystemHandle** pVsx, const
char* ipAddress, const char* pluginName)

```



```

01517 {
01518     if (vsx_InitTcpSensor_ptr == NULL)
01519     {
01520         const char_t* methodName = DNNE_STR("InitTcpSensor");
01521         vsx_InitTcpSensor_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle** pVsx, const char*
01522         ipAddress, const char* pluginName))get_fast_callable_managed_function(tl_name, methodName);
01523     }
01524     return vsx_InitTcpSensor_ptr(pVsx, ipAddress, pluginName);
01525 }
01526 // Computed from PF.VsxCProtocolDriver.Wrapper.VsxCExports.InitSerialSensor
01527 static VsxCStatusCode (DNNE_CALLTYPE* vsx_InitSerialSensor_ptr)(VsxCSystemHandle** pVsx, const char*
01528 serialPort, int32_t baudrate, const char* sensorType, VsxCSerialConnectionType connectionType, const
01529 char* pluginName);
01528 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_InitSerialSensor(VsxCSystemHandle** pVsx, const
01529 char* serialPort, int32_t baudrate, const char* sensorType, VsxCSerialConnectionType connectionType,
01530 const char* pluginName)
01529 {
01530     if (vsx_InitSerialSensor_ptr == NULL)
01531     {
01532         const char_t* methodName = DNNE_STR("InitSerialSensor");
01533         vsx_InitSerialSensor_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle** pVsx, const char*
01534         serialPort, int32_t baudrate, const char* sensorType, VsxCSerialConnectionType connectionType, const
01535         char* pluginName))get_fast_callable_managed_function(tl_name, methodName);
01536     }
01537     return vsx_InitSerialSensor_ptr(pVsx, serialPort, baudrate, sensorType, connectionType,
01538     pluginName);
01539 }
01540 // Computed from PF.VsxCProtocolDriver.Wrapper.VsxCExports.ReleaseSensor
01541 static VsxCStatusCode (DNNE_CALLTYPE* vsx_ReleaseSensor_ptr)(VsxCSystemHandle** vsx);
01542 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_ReleaseSensor(VsxCSystemHandle** vsx)
01543 {
01544     if (vsx_ReleaseSensor_ptr == NULL)
01545     {
01546         const char_t* methodName = DNNE_STR("ReleaseSensor");
01547         vsx_ReleaseSensor_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle**
01548         vsx))get_fast_callable_managed_function(tl_name, methodName);
01549     }
01550     return vsx_ReleaseSensor_ptr(vsx);
01551 }
01552 // Computed from PF.VsxCProtocolDriver.Wrapper.VsxCExports.ReConnectTcpDevice
01553 static VsxCStatusCode (DNNE_CALLTYPE* vsx_ReConnectTcpDevice_ptr)(VsxCSystemHandle* vsx, const char*
01554 ipAddress);
01555 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_ReConnectTcpDevice(VsxCSystemHandle* vsx, const
01556 char* ipAddress)
01557 {
01558     if (vsx_ReConnectTcpDevice_ptr == NULL)
01559     {
01560         const char_t* methodName = DNNE_STR("ReConnectTcpDevice");
01561         vsx_ReConnectTcpDevice_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle* vsx, const char*
01562         ipAddress))get_fast_callable_managed_function(tl_name, methodName);
01563     }
01564     return vsx_ReConnectTcpDevice_ptr(vsx, ipAddress);
01565 }
01566 // Computed from PF.VsxCProtocolDriver.Wrapper.VsxCExports.ReConnectAndLoginTcpDevice
01567 static VsxCStatusCode (DNNE_CALLTYPE* vsx_ReConnectAndLoginTcpDevice_ptr)(VsxCSystemHandle* vsx, const
01568 char* ipAddress, const char* username, const char* password);
01569 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_ReConnectAndLoginTcpDevice(VsxCSystemHandle*
01570 vsx, const char* ipAddress, const char* username, const char* password)
01571 {
01572     if (vsx_ReConnectAndLoginTcpDevice_ptr == NULL)
01573     {
01574         const char_t* methodName = DNNE_STR("ReConnectAndLoginTcpDevice");
01575         vsx_ReConnectAndLoginTcpDevice_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle* vsx,
01576         const char* ipAddress, const char* username, const char*
01577         password))get_fast_callable_managed_function(tl_name, methodName);
01578     }
01579     return vsx_ReConnectAndLoginTcpDevice_ptr(vsx, ipAddress, username, password);
01580 }
01581 // Computed from PF.VsxCProtocolDriver.Wrapper.VsxCExports.ReConnectSerialDevice
01582 static VsxCStatusCode (DNNE_CALLTYPE* vsx_ReConnectSerialDevice_ptr)(VsxCSystemHandle* vsx, const char*
01583 serialPort, int32_t baudrate, VsxCSerialConnectionType connectionType);
01584 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_ReConnectSerialDevice(VsxCSystemHandle* vsx,
01585 const char* serialPort, int32_t baudrate, VsxCSerialConnectionType connectionType)
01586 {
01587     if (vsx_ReConnectSerialDevice_ptr == NULL)
01588     {
01589         const char_t* methodName = DNNE_STR("ReConnectSerialDevice");
01590         vsx_ReConnectSerialDevice_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle* vsx, const
01591         char* serialPort, int32_t baudrate, VsxCSerialConnectionType
01592         connectionType))get_fast_callable_managed_function(tl_name, methodName);
01593     }
01594     return vsx_ReConnectSerialDevice_ptr(vsx, serialPort, baudrate, connectionType);
01595 }

```

```

01584 }
01585
01586 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Connect
01587 static VsxStatusCode (DNNE_CALLTYPE* vsx_Connect_ptr) (VsxSystemHandle* vsx);
01588 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Connect (VsxSystemHandle* vsx)
01589 {
01590     if (vsx_Connect_ptr == NULL)
01591     {
01592         const char_t* methodName = DNNE_STR("Connect");
01593         vsx_Connect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle*
01594 vsx) get_fast_callable_managed_function(tl_name, methodName);
01595     }
01596     return vsx_Connect_ptr(vsx);
01597 }
01598
01599 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectEx
01600 static VsxStatusCode (DNNE_CALLTYPE* vsx_ConnectEx_ptr) (VsxSystemHandle* vsx, int32_t timeout_ms);
01601 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectEx (VsxSystemHandle* vsx, int32_t
01602 timeout_ms)
01603 {
01604     if (vsx_ConnectEx_ptr == NULL)
01605     {
01606         const char_t* methodName = DNNE_STR("ConnectEx");
01607         vsx_ConnectEx_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, int32_t
01608 timeout_ms) get_fast_callable_managed_function(tl_name, methodName);
01609     }
01610     return vsx_ConnectEx_ptr(vsx, timeout_ms);
01611 }
01612
01613 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectAndLogin
01614 static VsxStatusCode (DNNE_CALLTYPE* vsx_ConnectAndLogin_ptr) (VsxSystemHandle* vsx, const char*
01615 username, const char* password);
01616 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectAndLogin (VsxSystemHandle* vsx, const
01617 char* username, const char* password)
01618 {
01619     if (vsx_ConnectAndLogin_ptr == NULL)
01620     {
01621         const char_t* methodName = DNNE_STR("ConnectAndLogin");
01622         vsx_ConnectAndLogin_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char*
01623 username, const char* password) get_fast_callable_managed_function(tl_name, methodName);
01624     }
01625     return vsx_ConnectAndLogin_ptr(vsx, username, password);
01626 }
01627
01628 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.ConnectExAndLogin
01629 static VsxStatusCode (DNNE_CALLTYPE* vsx_ConnectExAndLogin_ptr) (VsxSystemHandle* vsx, const char*
01630 username, const char* password, int32_t timeout_ms);
01631 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ConnectExAndLogin (VsxSystemHandle* vsx, const
01632 char* username, const char* password, int32_t timeout_ms)
01633 {
01634     if (vsx_ConnectExAndLogin_ptr == NULL)
01635     {
01636         const char_t* methodName = DNNE_STR("ConnectExAndLogin");
01637         vsx_ConnectExAndLogin_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char*
01638 username, const char* password, int32_t timeout_ms) get_fast_callable_managed_function(tl_name,
01639 methodName);
01640     }
01641     return vsx_ConnectExAndLogin_ptr(vsx, username, password, timeout_ms);
01642 }
01643
01644 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Login
01645 static VsxStatusCode (DNNE_CALLTYPE* vsx_Login_ptr) (VsxSystemHandle* vsx, const char* username, const
01646 char* password);
01647 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Login (VsxSystemHandle* vsx, const char*
01648 username, const char* password)
01649 {
01650     if (vsx_Login_ptr == NULL)
01651     {
01652         const char_t* methodName = DNNE_STR("Login");
01653         vsx_Login_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char* username,
01654 const char* password) get_fast_callable_managed_function(tl_name, methodName);
01655     }
01656     return vsx_Login_ptr(vsx, username, password);
01657 }
01658
01659 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Logout
01660 static VsxStatusCode (DNNE_CALLTYPE* vsx_Logout_ptr) (VsxSystemHandle* vsx);
01661 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Logout (VsxSystemHandle* vsx)
01662 {
01663     if (vsx_Logout_ptr == NULL)
01664     {
01665         const char_t* methodName = DNNE_STR("Logout");
01666         vsx_Logout_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle*
01667 vsx) get_fast_callable_managed_function(tl_name, methodName);
01668     }
01669     return vsx_Logout_ptr(vsx);
01670 }

```

```

01657
01658 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetPassword
01659 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetPassword_ptr)(VsxSystemHandle* vsx, const char*
authorizationUsername, const char* authorizationPassword, const char* username, const char* password);
01660 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetPassword(VsxSystemHandle* vsx, const char*
authorizationUsername, const char* authorizationPassword, const char* username, const char* password)
01661 {
01662     if (vsx_SetPassword_ptr == NULL)
01663     {
01664         const char_t* methodName = DNNE_STR("SetPassword");
01665         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(tl_name, methodName);
01666     }
01667     return vsx_SetPassword_ptr(vsx, authorizationUsername, authorizationPassword, username, password);
01668 }
01669
01670 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetConnected
01671 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetConnected_ptr)(VsxSystemHandle* vsx, int32_t* result);
01672 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetConnected(VsxSystemHandle* vsx, int32_t*
result)
01673 {
01674     if (vsx_GetConnected_ptr == NULL)
01675     {
01676         const char_t* methodName = DNNE_STR("GetConnected");
01677         vsx_GetConnected_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, int32_t*
result) get_fast_callable_managed_function(tl_name, methodName);
01678     }
01679     return vsx_GetConnected_ptr(vsx, result);
01680 }
01681
01682 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.Disconnect
01683 static VsxStatusCode (DNNE_CALLTYPE* vsx_Disconnect_ptr)(VsxSystemHandle* vsx);
01684 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_Disconnect(VsxSystemHandle* vsx)
01685 {
01686     if (vsx_Disconnect_ptr == NULL)
01687     {
01688         const char_t* methodName = DNNE_STR("Disconnect");
01689         vsx_Disconnect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle*
vsx) get_fast_callable_managed_function(tl_name, methodName);
01690     }
01691     return vsx_Disconnect_ptr(vsx);
01692 }
01693
01694 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnDisconnect
01695 static VsxStatusCode (DNNE_CALLTYPE* vsx_RegisterOnDisconnect_ptr)(VsxSystemHandle* vsx,
vsx_OnDisconnect fptr);
01696 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_RegisterOnDisconnect(VsxSystemHandle* vsx,
vsx_OnDisconnect fptr)
01697 {
01698     if (vsx_RegisterOnDisconnect_ptr == NULL)
01699     {
01700         const char_t* methodName = DNNE_STR("RegisterOnDisconnect");
01701         vsx_RegisterOnDisconnect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx,
vsx_OnDisconnect fptr) get_fast_callable_managed_function(tl_name, methodName);
01702     }
01703     return vsx_RegisterOnDisconnect_ptr(vsx, fptr);
01704 }
01705
01706 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnDisconnect
01707 static VsxStatusCode (DNNE_CALLTYPE* vsx_DeregisterOnDisconnect_ptr)(VsxSystemHandle* vsx);
01708 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DeregisterOnDisconnect(VsxSystemHandle* vsx)
01709 {
01710     if (vsx_DeregisterOnDisconnect_ptr == NULL)
01711     {
01712         const char_t* methodName = DNNE_STR("DeregisterOnDisconnect");
01713         vsx_DeregisterOnDisconnect_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle*
vsx) get_fast_callable_managed_function(tl_name, methodName);
01714     }
01715     return vsx_DeregisterOnDisconnect_ptr(vsx);
01716 }
01717
01718 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.RegisterOnSessionMessageReceived
01719 static VsxStatusCode (DNNE_CALLTYPE* vsx_RegisterOnSessionMessageReceived_ptr)(VsxSystemHandle* vsx,
vsx_OnSessionMessageReceived fptr);
01720 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_RegisterOnSessionMessageReceived(VsxSystemHandle* vsx, vsx_OnSessionMessageReceived fptr)
01721 {
01722     if (vsx_RegisterOnSessionMessageReceived_ptr == NULL)
01723     {
01724         const char_t* methodName = DNNE_STR("RegisterOnSessionMessageReceived");
01725         vsx_RegisterOnSessionMessageReceived_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle*
vsx, vsx_OnSessionMessageReceived fptr) get_fast_callable_managed_function(tl_name, methodName);
01726     }
01727     return vsx_RegisterOnSessionMessageReceived_ptr(vsx, fptr);
01728 }
01729

```

```

01730 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.DeregisterOnSessionMessageReceived
01731 static VsxStatusCode (DNNE_CALLTYPE* vsx_DeregisterOnSessionMessageReceived_ptr) (VsxSystemHandle*
vsx);
01732 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_DeregisterOnSessionMessageReceived(VsxSystemHandle* vsx)
01733 {
01734     if (vsx_DeregisterOnSessionMessageReceived_ptr == NULL)
01735     {
01736         const char_t* methodName = DNNE_STR("DeregisterOnSessionMessageReceived");
01737         vsx_DeregisterOnSessionMessageReceived_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxSystemHandle*
vsx))get_fast_callable_managed_function(tl_name, methodName);
01738     }
01739     return vsx_DeregisterOnSessionMessageReceived_ptr(vsx);
01740 }
01741
01742 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendSessionKeepAlive
01743 static VsxStatusCode (DNNE_CALLTYPE* vsx_SendSessionKeepAlive_ptr) (VsxSystemHandle* vsx);
01744 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendSessionKeepAlive(VsxSystemHandle* vsx)
01745 {
01746     if (vsx_SendSessionKeepAlive_ptr == NULL)
01747     {
01748         const char_t* methodName = DNNE_STR("SendSessionKeepAlive");
01749         vsx_SendSessionKeepAlive_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxSystemHandle*
vsx))get_fast_callable_managed_function(tl_name, methodName);
01750     }
01751     return vsx_SendSessionKeepAlive_ptr(vsx);
01752 }
01753
01754 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystem
01755 static VsxStatusCode (DNNE_CALLTYPE* vsx_TestSystem_ptr) (VsxSystemHandle* vsx, const char* command,
const char* inputValue, const char** outputValue, int32_t* status);
01756 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_TestSystem(VsxSystemHandle* vsx, const char*
command, const char* inputValue, const char** outputValue, int32_t* status)
01757 {
01758     if (vsx_TestSystem_ptr == NULL)
01759     {
01760         const char_t* methodName = DNNE_STR("TestSystem");
01761         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(tl_name, methodName);
01762     }
01763     return vsx_TestSystem_ptr(vsx, command, inputValue, outputValue, status);
01764 }
01765
01766 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.TestSystemEx
01767 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);
01768 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)
01769 {
01770     if (vsx_TestSystemEx_ptr == NULL)
01771     {
01772         const char_t* methodName = DNNE_STR("TestSystemEx");
01773         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(tl_name, methodName);
01774     }
01775     return vsx_TestSystemEx_ptr(vsx, command, inputValue, outputValue, status, timeout_ms);
01776 }
01777
01778 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.GetWaitTimeout
01779 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetWaitTimeout_ptr) (VsxSystemHandle* vsx, int32_t*
timeout_ms);
01780 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetWaitTimeout(VsxSystemHandle* vsx, int32_t*
timeout_ms)
01781 {
01782     if (vsx_GetWaitTimeout_ptr == NULL)
01783     {
01784         const char_t* methodName = DNNE_STR("GetWaitTimeout");
01785         vsx_GetWaitTimeout_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxSystemHandle* vsx, int32_t*
timeout_ms))get_fast_callable_managed_function(tl_name, methodName);
01786     }
01787     return vsx_GetWaitTimeout_ptr(vsx, timeout_ms);
01788 }
01789
01790 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetWaitTimeout
01791 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetWaitTimeout_ptr) (VsxSystemHandle* vsx, int32_t
timeout_ms);
01792 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetWaitTimeout(VsxSystemHandle* vsx, int32_t
timeout_ms)
01793 {
01794     if (vsx_SetWaitTimeout_ptr == NULL)
01795     {
01796         const char_t* methodName = DNNE_STR("SetWaitTimeout");
01797         vsx_SetWaitTimeout_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxSystemHandle* vsx, int32_t
timeout_ms))get_fast_callable_managed_function(tl_name, methodName);
01798     }

```

```

01799     return vsx_SetWaitTimeout_ptr(vsx, timeout_ms);
01800 }
01801
01802 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.UploadData
01803 static VsxStatusCode (DNNE_CALLTYPE* vsx_UploadData_ptr) (VsxSystemHandle* vsx, const char* fileName);
01804 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadData(VsxSystemHandle* vsx, const char*
fileName)
01805 {
01806     if (vsx_UploadData_ptr == NULL)
01807     {
01808         const char_t* methodName = DNNE_STR("UploadData");
01809         vsx_UploadData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char*
fileName) get_fast_callable_managed_function(tl_name, methodName);
01810     }
01811     return vsx_UploadData_ptr(vsx, fileName);
01812 }
01813
01814 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendFirmware
01815 static VsxStatusCode (DNNE_CALLTYPE* vsx_SendFirmware_ptr) (VsxSystemHandle* vsx, const char*
fileName);
01816 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendFirmware(VsxSystemHandle* vsx, const char*
fileName)
01817 {
01818     if (vsx_SendFirmware_ptr == NULL)
01819     {
01820         const char_t* methodName = DNNE_STR("SendFirmware");
01821         vsx_SendFirmware_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char*
fileName) get_fast_callable_managed_function(tl_name, methodName);
01822     }
01823     return vsx_SendFirmware_ptr(vsx, fileName);
01824 }
01825
01826 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SendXmlDataMessage
01827 static VsxStatusCode (DNNE_CALLTYPE* vsx_SendXmlDataMessage_ptr) (VsxSystemHandle* vsx, const char*
xmlCommand);
01828 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SendXmlDataMessage(VsxSystemHandle* vsx, const
char* xmlCommand)
01829 {
01830     if (vsx_SendXmlDataMessage_ptr == NULL)
01831     {
01832         const char_t* methodName = DNNE_STR("SendXmlDataMessage");
01833         vsx_SendXmlDataMessage_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char*
xmlCommand) get_fast_callable_managed_function(tl_name, methodName);
01834     }
01835     return vsx_SendXmlDataMessage_ptr(vsx, xmlCommand);
01836 }
01837
01838 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettings
01839 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetNetworkSettings_ptr) (VsxSystemHandle* vsx, const char*
ipAddress, const char* networkMask, const char* gateway);
01840 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettings(VsxSystemHandle* vsx, const
char* ipAddress, const char* networkMask, const char* gateway)
01841 {
01842     if (vsx_SetNetworkSettings_ptr == NULL)
01843     {
01844         const char_t* methodName = DNNE_STR("SetNetworkSettings");
01845         vsx_SetNetworkSettings_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const char*
ipAddress, const char* networkMask, const char* gateway) get_fast_callable_managed_function(tl_name,
methodName);
01846     }
01847     return vsx_SetNetworkSettings_ptr(vsx, ipAddress, networkMask, gateway);
01848 }
01849
01850 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExports.SetNetworkSettingsViaUdp
01851 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetNetworkSettingsViaUdp_ptr) (const char* macAddress, const
char* ipAddress, const char* networkMask, const char* gateway);
01852 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetNetworkSettingsViaUdp(const char*
macAddress, const char* ipAddress, const char* networkMask, const char* gateway)
01853 {
01854     if (vsx_SetNetworkSettingsViaUdp_ptr == NULL)
01855     {
01856         const char_t* methodName = DNNE_STR("SetNetworkSettingsViaUdp");
01857         vsx_SetNetworkSettingsViaUdp_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (const char* macAddress,
const char* ipAddress, const char* networkMask, const char*
gateway) get_fast_callable_managed_function(tl_name, methodName);
01858     }
01859     return vsx_SetNetworkSettingsViaUdp_ptr(macAddress, ipAddress, networkMask, gateway);
01860 }
01861
01862 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ResetDynamicContainerGrabber
01863 static VsxStatusCode (DNNE_CALLTYPE* vsx_ResetDynamicContainerGrabber_ptr) (VsxSystemHandle* vsx,
int32_t bufferSize, VsxStrategy strategy);
01864 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetDynamicContainerGrabber(VsxSystemHandle*
vsx, int32_t bufferSize, VsxStrategy strategy)
01865 {
01866     if (vsx_ResetDynamicContainerGrabber_ptr == NULL)
01867     {

```

```

01868     const char_t* methodName = DNNE_STR("ResetDynamicContainerGrabber");
01869     vsx_ResetDynamicContainerGrabber_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle* vsx,
int32_t bufferSize, VsxCStrategy strategy))get_fast_callable_managed_function(t2_name, methodName);
01870 }
01871 return vsx_ResetDynamicContainerGrabber_ptr(vsx, bufferSize, strategy);
01872 }
01873
01874 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDataContainer
01875 static VsxCStatusCode (DNNE_CALLTYPE* vsx_GetDataContainer_ptr) (VsxCSystemHandle* vsx,
VsxCDataContainerHandle** pDch, int32_t timeout_ms);
01876 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_GetDataContainer(VsxCSystemHandle* vsx,
VsxCDataContainerHandle** pDch, int32_t timeout_ms)
01877 {
01878     if (vsx_GetDataContainer_ptr == NULL)
01879     {
01880         const char_t* methodName = DNNE_STR("GetDataContainer");
01881         vsx_GetDataContainer_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle* vsx,
VsxCDataContainerHandle** pDch, int32_t timeout_ms))get_fast_callable_managed_function(t2_name,
methodName);
01882     }
01883     return vsx_GetDataContainer_ptr(vsx, pDch, timeout_ms);
01884 }
01885
01886 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCachedContainer
01887 static VsxCStatusCode (DNNE_CALLTYPE* vsx_GetCachedContainer_ptr) (VsxCSystemHandle* vsx,
VsxCDataContainerHandle** pDch, int32_t position);
01888 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_GetCachedContainer(VsxCSystemHandle* vsx,
VsxCDataContainerHandle** pDch, int32_t position)
01889 {
01890     if (vsx_GetCachedContainer_ptr == NULL)
01891     {
01892         const char_t* methodName = DNNE_STR("GetCachedContainer");
01893         vsx_GetCachedContainer_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCSystemHandle* vsx,
VsxCDataContainerHandle** pDch, int32_t position))get_fast_callable_managed_function(t2_name,
methodName);
01894     }
01895     return vsx_GetCachedContainer_ptr(vsx, pDch, position);
01896 }
01897
01898 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDataContainer
01899 static VsxCStatusCode (DNNE_CALLTYPE* vsx_ReleaseDataContainer_ptr) (VsxCDataContainerHandle** dch);
01900 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_ReleaseDataContainer(VsxCDataContainerHandle**
dch)
01901 {
01902     if (vsx_ReleaseDataContainer_ptr == NULL)
01903     {
01904         const char_t* methodName = DNNE_STR("ReleaseDataContainer");
01905         vsx_ReleaseDataContainer_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCDataContainerHandle**
dch))get_fast_callable_managed_function(t2_name, methodName);
01906     }
01907     return vsx_ReleaseDataContainer_ptr(dch);
01908 }
01909
01910 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.SaveData
01911 static VsxCStatusCode (DNNE_CALLTYPE* vsx_SaveData_ptr) (VsxCDataContainerHandle* dch, const char* tag,
const char* fileName);
01912 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_SaveData(VsxCDataContainerHandle* dch, const
char* tag, const char* fileName)
01913 {
01914     if (vsx_SaveData_ptr == NULL)
01915     {
01916         const char_t* methodName = DNNE_STR("SaveData");
01917         vsx_SaveData_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCDataContainerHandle* dch, const char*
tag, const char* fileName))get_fast_callable_managed_function(t2_name, methodName);
01918     }
01919     return vsx_SaveData_ptr(dch, tag, fileName);
01920 }
01921
01922 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.Save3DPointCloudData
01923 static VsxCStatusCode (DNNE_CALLTYPE* vsx_Save3DPointCloudData_ptr) (VsxCDataContainerHandle* dch, const
char* point_x_Id, const char* point_y_Id, const char* point_z_Id, const char* fileName);
01924 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_Save3DPointCloudData(VsxCDataContainerHandle*
dch, const char* point_x_Id, const char* point_y_Id, const char* point_z_Id, const char* fileName)
01925 {
01926     if (vsx_Save3DPointCloudData_ptr == NULL)
01927     {
01928         const char_t* methodName = DNNE_STR("Save3DPointCloudData");
01929         vsx_Save3DPointCloudData_ptr = (VsxCStatusCode(DNNE_CALLTYPE*)(VsxCDataContainerHandle* 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);
01930     }
01931     return vsx_Save3DPointCloudData_ptr(dch, point_x_Id, point_y_Id, point_z_Id, fileName);
01932 }
01933
01934 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetImage
01935 static VsxCStatusCode (DNNE_CALLTYPE* vsx_GetImage_ptr) (VsxCDataContainerHandle* dch, const char* tag,
VsxCImage** imageData);

```



```

01936 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetImage(VsxDataContainerHandle* dch, const
char* tag, VsxImage** imageData)
01937 {
01938     if (vsx_GetImage_ptr == NULL)
01939     {
01940         const char_t* methodName = DNNE_STR("GetImage");
01941         vsx_GetImage_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch, const char*
tag, VsxImage** imageData))get_fast_callable_managed_function(t2_name, methodName);
01942     }
01943     return vsx_GetImage_ptr(dch, tag, imageData);
01944 }
01945
01946 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseImage
01947 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseImage_ptr) (VsxImage** pImage);
01948 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseImage(VsxImage** pImage)
01949 {
01950     if (vsx_ReleaseImage_ptr == NULL)
01951     {
01952         const char_t* methodName = DNNE_STR("ReleaseImage");
01953         vsx_ReleaseImage_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxImage**
pImage))get_fast_callable_managed_function(t2_name, methodName);
01954     }
01955     return vsx_ReleaseImage_ptr(pImage);
01956 }
01957
01958 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetLine
01959 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLine_ptr) (VsxDataContainerHandle* dch, const char* tag,
VsxLineData** data);
01960 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLine(VsxDataContainerHandle* dch, const
char* tag, VsxLineData** data)
01961 {
01962     if (vsx_GetLine_ptr == NULL)
01963     {
01964         const char_t* methodName = DNNE_STR("GetLine");
01965         vsx_GetLine_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch, const char* tag,
VsxLineData** data))get_fast_callable_managed_function(t2_name, methodName);
01966     }
01967     return vsx_GetLine_ptr(dch, tag, data);
01968 }
01969
01970 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseLine
01971 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseLine_ptr) (VsxLineData** pLineData);
01972 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseLine(VsxLineData** pLineData)
01973 {
01974     if (vsx_ReleaseLine_ptr == NULL)
01975     {
01976         const char_t* methodName = DNNE_STR("ReleaseLine");
01977         vsx_ReleaseLine_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxLineData**
pLineData))get_fast_callable_managed_function(t2_name, methodName);
01978     }
01979     return vsx_ReleaseLine_ptr(pLineData);
01980 }
01981
01982 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDisparityDescriptor2
01983 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDisparityDescriptor2_ptr) (VsxDataContainerHandle* dch,
const char* tag, VsxDisparityDescriptor2** data);
01984 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDisparityDescriptor2(VsxDataContainerHandle*
dch, const char* tag, VsxDisparityDescriptor2** data)
01985 {
01986     if (vsx_GetDisparityDescriptor2_ptr == NULL)
01987     {
01988         const char_t* methodName = DNNE_STR("GetDisparityDescriptor2");
01989         vsx_GetDisparityDescriptor2_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch,
const char* tag, VsxDisparityDescriptor2** data))get_fast_callable_managed_function(t2_name,
methodName);
01990     }
01991     return vsx_GetDisparityDescriptor2_ptr(dch, tag, data);
01992 }
01993
01994 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseDisparityDescriptor2
01995 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDisparityDescriptor2_ptr) (VsxDisparityDescriptor2**
pData);
01996 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_ReleaseDisparityDescriptor2(VsxDisparityDescriptor2** pData)
01997 {
01998     if (vsx_ReleaseDisparityDescriptor2_ptr == NULL)
01999     {
02000         const char_t* methodName = DNNE_STR("ReleaseDisparityDescriptor2");
02001         vsx_ReleaseDisparityDescriptor2_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDisparityDescriptor2**
pData))get_fast_callable_managed_function(t2_name, methodName);
02002     }
02003     return vsx_ReleaseDisparityDescriptor2_ptr(pData);
02004 }
02005
02006 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTransformation
02007 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetTransformation_ptr) (VsxDataContainerHandle* dch, const
char* tag, VsxTransformation** data);

```

```

02008 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTransformation(VsxDataContainerHandle* dch,
    const char* tag, VsxTransformation** data)
02009 {
02010     if (vsx_GetTransformation_ptr == NULL)
02011     {
02012         const char_t* methodName = DNNE_STR("GetTransformation");
02013         vsx_GetTransformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch, const
    char* tag, VsxTransformation** data))get_fast_callable_managed_function(t2_name, methodName);
02014     }
02015     return vsx_GetTransformation_ptr(dch, tag, data);
02016 }
02017
02018 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTransformation
02019 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseTransformation_ptr)(VsxTransformation** pData);
02020 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTransformation(VsxTransformation**
    pData)
02021 {
02022     if (vsx_ReleaseTransformation_ptr == NULL)
02023     {
02024         const char_t* methodName = DNNE_STR("ReleaseTransformation");
02025         vsx_ReleaseTransformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxTransformation**
    pData))get_fast_callable_managed_function(t2_name, methodName);
02026     }
02027     return vsx_ReleaseTransformation_ptr(pData);
02028 }
02029
02030 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetCaptureInformation
02031 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetCaptureInformation_ptr)(VsxDataContainerHandle* dch, const
    char* tag, VsxCaptureInformation** data);
02032 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetCaptureInformation(VsxDataContainerHandle*
    dch, const char* tag, VsxCaptureInformation** data)
02033 {
02034     if (vsx_GetCaptureInformation_ptr == NULL)
02035     {
02036         const char_t* methodName = DNNE_STR("GetCaptureInformation");
02037         vsx_GetCaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch,
    const char* tag, VsxCaptureInformation** data))get_fast_callable_managed_function(t2_name,
    methodName);
02038     }
02039     return vsx_GetCaptureInformation_ptr(dch, tag, data);
02040 }
02041
02042 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseCaptureInformation
02043 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseCaptureInformation_ptr)(VsxCaptureInformation**
    pData);
02044 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
    vsx_ReleaseCaptureInformation(VsxCaptureInformation** pData)
02045 {
02046     if (vsx_ReleaseCaptureInformation_ptr == NULL)
02047     {
02048         const char_t* methodName = DNNE_STR("ReleaseCaptureInformation");
02049         vsx_ReleaseCaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxCaptureInformation**
    pData))get_fast_callable_managed_function(t2_name, methodName);
02050     }
02051     return vsx_ReleaseCaptureInformation_ptr(pData);
02052 }
02053
02054 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2CaptureInformation
02055 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetOlr2CaptureInformation_ptr)(VsxDataContainerHandle* dch,
    const char* tag, VsxOlr2CaptureInformation** data);
02056 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
    vsx_GetOlr2CaptureInformation(VsxDataContainerHandle* dch, const char* tag,
    VsxOlr2CaptureInformation** data)
02057 {
02058     if (vsx_GetOlr2CaptureInformation_ptr == NULL)
02059     {
02060         const char_t* methodName = DNNE_STR("GetOlr2CaptureInformation");
02061         vsx_GetOlr2CaptureInformation_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle*
    dch, const char* tag, VsxOlr2CaptureInformation** data))get_fast_callable_managed_function(t2_name,
    methodName);
02062     }
02063     return vsx_GetOlr2CaptureInformation_ptr(dch, tag, data);
02064 }
02065
02066 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2CaptureInformation
02067 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseOlr2CaptureInformation_ptr)(VsxOlr2CaptureInformation**
    pData);
02068 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
    vsx_ReleaseOlr2CaptureInformation(VsxOlr2CaptureInformation** pData)
02069 {
02070     if (vsx_ReleaseOlr2CaptureInformation_ptr == NULL)
02071     {
02072         const char_t* methodName = DNNE_STR("ReleaseOlr2CaptureInformation");
02073         vsx_ReleaseOlr2CaptureInformation_ptr =
    (VsxStatusCode(DNNE_CALLTYPE*) (VsxOlr2CaptureInformation**
    pData))get_fast_callable_managed_function(t2_name, methodName);
02074     }

```



```

02075     return vsx_ReleaseOlr2CaptureInformation_ptr(pData);
02076 }
02077
02078 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetOlr2ModbusData
02079 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetOlr2ModbusData_ptr) (VsxDataContainerHandle* dch, const
char* tag, VsxOlr2ModbusData** data);
02080 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetOlr2ModbusData(VsxDataContainerHandle* dch,
const char* tag, VsxOlr2ModbusData** data)
02081 {
02082     if (vsx_GetOlr2ModbusData_ptr == NULL)
02083     {
02084         const char_t* methodName = DNNE_STR("GetOlr2ModbusData");
02085         vsx_GetOlr2ModbusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxDataContainerHandle* dch, const
char* tag, VsxOlr2ModbusData** data) get_fast_callable_managed_function(t2_name, methodName);
02086     }
02087     return vsx_GetOlr2ModbusData_ptr(dch, tag, data);
02088 }
02089
02090 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseOlr2ModbusData
02091 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseOlr2ModbusData_ptr) (VsxOlr2ModbusData** pData);
02092 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseOlr2ModbusData(VsxOlr2ModbusData**
pData)
02093 {
02094     if (vsx_ReleaseOlr2ModbusData_ptr == NULL)
02095     {
02096         const char_t* methodName = DNNE_STR("ReleaseOlr2ModbusData");
02097         vsx_ReleaseOlr2ModbusData_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxOlr2ModbusData**
pData) get_fast_callable_managed_function(t2_name, methodName);
02098     }
02099     return vsx_ReleaseOlr2ModbusData_ptr(pData);
02100 }
02101
02102 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetTagList
02103 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetTagList_ptr) (VsxDataContainerHandle* dch, VsxTagList**
tagList);
02104 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetTagList(VsxDataContainerHandle* dch,
VsxTagList** tagList)
02105 {
02106     if (vsx_GetTagList_ptr == NULL)
02107     {
02108         const char_t* methodName = DNNE_STR("GetTagList");
02109         vsx_GetTagList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxDataContainerHandle* dch, VsxTagList**
tagList) get_fast_callable_managed_function(t2_name, methodName);
02110     }
02111     return vsx_GetTagList_ptr(dch, tagList);
02112 }
02113
02114 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.ReleaseTagList
02115 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseTagList_ptr) (VsxTagList** pTagList);
02116 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseTagList(VsxTagList** pTagList)
02117 {
02118     if (vsx_ReleaseTagList_ptr == NULL)
02119     {
02120         const char_t* methodName = DNNE_STR("ReleaseTagList");
02121         vsx_ReleaseTagList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxTagList**
pTagList) get_fast_callable_managed_function(t2_name, methodName);
02122     }
02123     return vsx_ReleaseTagList_ptr(pTagList);
02124 }
02125
02126 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetMissingContainerFramesCounter
02127 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetMissingContainerFramesCounter_ptr) (VsxSystemHandle* vsx,
int32_t* result);
02128 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_GetMissingContainerFramesCounter(VsxSystemHandle* vsx, int32_t* result)
02129 {
02130     if (vsx_GetMissingContainerFramesCounter_ptr == NULL)
02131     {
02132         const char_t* methodName = DNNE_STR("GetMissingContainerFramesCounter");
02133         vsx_GetMissingContainerFramesCounter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle*
vsx, int32_t* result) get_fast_callable_managed_function(t2_name, methodName);
02134     }
02135     return vsx_GetMissingContainerFramesCounter_ptr(vsx, result);
02136 }
02137
02138 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetDynamicContainerQueueSize
02139 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDynamicContainerQueueSize_ptr) (VsxSystemHandle* vsx,
int32_t* result);
02140 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDynamicContainerQueueSize(VsxSystemHandle*
vsx, int32_t* result)
02141 {
02142     if (vsx_GetDynamicContainerQueueSize_ptr == NULL)
02143     {
02144         const char_t* methodName = DNNE_STR("GetDynamicContainerQueueSize");
02145         vsx_GetDynamicContainerQueueSize_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx,
int32_t* result) get_fast_callable_managed_function(t2_name, methodName);
02146     }

```

```

02147     return vsx_GetDynamicContainerQueueSize_ptr(vsx, result);
02148 }
02149
02150 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDataContainer.GetNumberOfCachedContainers
02151 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetNumberOfCachedContainers_ptr) (VsxSystemHandle* vsx,
02152 int32_t* result);
02153 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetNumberOfCachedContainers (VsxSystemHandle*
02154 vsx, int32_t* result)
02155 {
02156     if (vsx_GetNumberOfCachedContainers_ptr == NULL)
02157     {
02158         const char_t* methodName = DNNE_STR("GetNumberOfCachedContainers");
02159         vsx_GetNumberOfCachedContainers_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx,
02160 int32_t* result))get_fast_callable_managed_function(t2_name, methodName);
02161     }
02162     return vsx_GetNumberOfCachedContainers_ptr(vsx, result);
02163 }
02164
02165 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetDeviceInformation
02166 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetDeviceInformation_ptr) (VsxSystemHandle* vsx, VsxDevice**
02167 deviceData);
02168 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetDeviceInformation (VsxSystemHandle* vsx,
02169 VsxDevice** deviceData)
02170 {
02171     if (vsx_GetDeviceInformation_ptr == NULL)
02172     {
02173         const char_t* methodName = DNNE_STR("GetDeviceInformation");
02174         vsx_GetDeviceInformation_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx,
02175 VsxDevice** deviceData))get_fast_callable_managed_function(t3_name, methodName);
02176     }
02177     return vsx_GetDeviceInformation_ptr(vsx, deviceData);
02178 }
02179
02180 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDevice
02181 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDevice_ptr) (VsxDevice** pDevice);
02182 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDevice (VsxDevice** pDevice)
02183 {
02184     if (vsx_ReleaseDevice_ptr == NULL)
02185     {
02186         const char_t* methodName = DNNE_STR("ReleaseDevice");
02187         vsx_ReleaseDevice_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxDevice**
02188 pDevice))get_fast_callable_managed_function(t3_name, methodName);
02189     }
02190     return vsx_ReleaseDevice_ptr(pDevice);
02191 }
02192
02193 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.GetUdpDeviceList
02194 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetUdpDeviceList_ptr) (VsxDeviceList** deviceListData);
02195 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetUdpDeviceList (VsxDeviceList**
02196 deviceListData)
02197 {
02198     if (vsx_GetUdpDeviceList_ptr == NULL)
02199     {
02200         const char_t* methodName = DNNE_STR("GetUdpDeviceList");
02201         vsx_GetUdpDeviceList_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxDeviceList**
02202 deviceListData))get_fast_callable_managed_function(t3_name, methodName);
02203     }
02204     return vsx_GetUdpDeviceList_ptr(deviceListData);
02205 }
02206
02207 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsDevice.ReleaseDeviceList
02208 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseDeviceList_ptr) (VsxDeviceList** pDeviceList);
02209 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseDeviceList (VsxDeviceList** pDeviceList)
02210 {
02211     if (vsx_ReleaseDeviceList_ptr == NULL)
02212     {
02213         const char_t* methodName = DNNE_STR("ReleaseDeviceList");
02214         vsx_ReleaseDeviceList_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxDeviceList**
02215 pDeviceList))get_fast_callable_managed_function(t3_name, methodName);
02216     }
02217     return vsx_ReleaseDeviceList_ptr(pDeviceList);
02218 }
02219
02220 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.ResetLogMessageGrabber
02221 static VsxStatusCode (DNNE_CALLTYPE* vsx_ResetLogMessageGrabber_ptr) (VsxSystemHandle* vsx, int32_t
02222 bufferSize, int32_t typeMask, VsxStrategy strategy);
02223 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ResetLogMessageGrabber (VsxSystemHandle* vsx,
02224 int32_t bufferSize, int32_t typeMask, VsxStrategy strategy)
02225 {
02226     if (vsx_ResetLogMessageGrabber_ptr == NULL)
02227     {
02228         const char_t* methodName = DNNE_STR("ResetLogMessageGrabber");
02229         vsx_ResetLogMessageGrabber_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx, int32_t
02230 bufferSize, int32_t typeMask, VsxStrategy strategy))get_fast_callable_managed_function(t4_name,
02231 methodName);
02232     }
02233     return vsx_ResetLogMessageGrabber_ptr(vsx, bufferSize, typeMask, strategy);
02234 }

```

```

02220 }
02221
02222 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessage
02223 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLogMessage_ptr) (VsxSystemHandle* vsx, const char** log,
    int32_t timeout_ms);
02224 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessage (VsxSystemHandle* vsx, const
    char** log, int32_t timeout_ms)
02225 {
02226     if (vsx_GetLogMessage_ptr == NULL)
02227     {
02228         const char_t* methodName = DNNE_STR("GetLogMessage");
02229         vsx_GetLogMessage_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx, const char** log,
    int32_t timeout_ms))get_fast_callable_managed_function(t4_name, methodName);
02230     }
02231     return vsx_GetLogMessage_ptr(vsx, log, timeout_ms);
02232 }
02233
02234 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetLogMessageQueueSize
02235 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetLogMessageQueueSize_ptr) (VsxSystemHandle* vsx, int32_t*
    result);
02236 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetLogMessageQueueSize (VsxSystemHandle* vsx,
    int32_t* result)
02237 {
02238     if (vsx_GetLogMessageQueueSize_ptr == NULL)
02239     {
02240         const char_t* methodName = DNNE_STR("GetLogMessageQueueSize");
02241         vsx_GetLogMessageQueueSize_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx, int32_t*
    result))get_fast_callable_managed_function(t4_name, methodName);
02242     }
02243     return vsx_GetLogMessageQueueSize_ptr(vsx, result);
02244 }
02245
02246 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsLog.GetMissingLogMessagesCounter
02247 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetMissingLogMessagesCounter_ptr) (VsxSystemHandle* vsx,
    int32_t* result);
02248 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetMissingLogMessagesCounter (VsxSystemHandle*
    vsx, int32_t* result)
02249 {
02250     if (vsx_GetMissingLogMessagesCounter_ptr == NULL)
02251     {
02252         const char_t* methodName = DNNE_STR("GetMissingLogMessagesCounter");
02253         vsx_GetMissingLogMessagesCounter_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx,
    int32_t* result))get_fast_callable_managed_function(t4_name, methodName);
02254     }
02255     return vsx_GetMissingLogMessagesCounter_ptr(vsx, result);
02256 }
02257
02258 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValue
02259 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterValue_ptr) (VsxSystemHandle* vsx, uint32_t
    settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId,
    const char* value);
02260 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)
02261 {
02262     if (vsx_SetSingleParameterValue_ptr == NULL)
02263     {
02264         const char_t* methodName = DNNE_STR("SetSingleParameterValue");
02265         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);
02266     }
02267     return vsx_SetSingleParameterValue_ptr(vsx, settingsVersion, configurationId,
    configurationVersion, parameterId, value);
02268 }
02269
02270 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueDouble
02271 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterDouble_ptr) (VsxSystemHandle* vsx,
    uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
    parameterId, double value);
02272 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble (VsxSystemHandle*
    vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
    parameterId, double value)
02273 {
02274     if (vsx_SetSingleParameterDouble_ptr == NULL)
02275     {
02276         const char_t* methodName = DNNE_STR("SetSingleParameterDouble");
02277         vsx_SetSingleParameterDouble_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);
02278     }
02279     return vsx_SetSingleParameterDouble_ptr(vsx, settingsVersion, configurationId,
    configurationVersion, parameterId, value);
02280 }
02281
02282 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SetSingleParameterValueInt32
02283 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterInt32_ptr) (VsxSystemHandle* vsx,

```

```

uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t value);
02284 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)
02285 {
02286     if (vsx_SetSingleParameterValueInt32_ptr == NULL)
02287     {
02288         const char_t* methodName = DNNE_STR("SetSingleParameterValueInt32");
02289         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);
02290     }
02291     return vsx_SetSingleParameterValueInt32_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
02292 }
02293
02294 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValue
02295 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameterValue_ptr)(VsxSystemHandle* vsx, uint32_t
settingsVersion, const char* configurationId, uint32_t configurationVersion, const char* parameterId,
const char** value);
02296 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)
02297 {
02298     if (vsx_GetSingleParameterValue_ptr == NULL)
02299     {
02300         const char_t* methodName = DNNE_STR("GetSingleParameterValue");
02301         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);
02302     }
02303     return vsx_GetSingleParameterValue_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
02304 }
02305
02306 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueDouble
02307 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameterValueDouble_ptr)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, double* value);
02308 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)
02309 {
02310     if (vsx_GetSingleParameterValueDouble_ptr == NULL)
02311     {
02312         const char_t* methodName = DNNE_STR("GetSingleParameterValueDouble");
02313         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);
02314     }
02315     return vsx_GetSingleParameterValueDouble_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
02316 }
02317
02318 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.GetSingleParameterValueInt32
02319 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameterInt32_ptr)(VsxSystemHandle* vsx,
uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t* value);
02320 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameterInt32(VsxSystemHandle*
vsx, uint32_t settingsVersion, const char* configurationId, uint32_t configurationVersion, const char*
parameterId, int32_t* value)
02321 {
02322     if (vsx_GetSingleParameterInt32_ptr == NULL)
02323     {
02324         const char_t* methodName = DNNE_STR("GetSingleParameterInt32");
02325         vsx_GetSingleParameterInt32_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);
02326     }
02327     return vsx_GetSingleParameterInt32_ptr(vsx, settingsVersion, configurationId,
configurationVersion, parameterId, value);
02328 }
02329
02330 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadDefaultParameterSetOnDevice
02331 static VsxStatusCode (DNNE_CALLTYPE* vsx_LoadDefaultParameterSetOnDevice_ptr)(VsxSystemHandle* vsx);
02332 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE
vsx_LoadDefaultParameterSetOnDevice(VsxSystemHandle* vsx)
02333 {
02334     if (vsx_LoadDefaultParameterSetOnDevice_ptr == NULL)
02335     {
02336         const char_t* methodName = DNNE_STR("LoadDefaultParameterSetOnDevice");
02337         vsx_LoadDefaultParameterSetOnDevice_ptr = (VsxStatusCode(DNNE_CALLTYPE*)(VsxSystemHandle*
vsx))get_fast_callable_managed_function(t5_name, methodName);
02338     }
02339     return vsx_LoadDefaultParameterSetOnDevice_ptr(vsx);
02340 }

```

```

02341
02342 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.LoadParameterSetOnDevice
02343 static VsxStatusCode (DNNE_CALLTYPE* vsx_LoadParameterSetOnDevice_ptr) (VsxSystemHandle* vsx);
02344 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_LoadParameterSetOnDevice (VsxSystemHandle* vsx)
02345 {
02346     if (vsx_LoadParameterSetOnDevice_ptr == NULL)
02347     {
02348         const char_t* methodName = DNNE_STR("LoadParameterSetOnDevice");
02349         vsx_LoadParameterSetOnDevice_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle*
vsx))get_fast_callable_managed_function(t5_name, methodName);
02350     }
02351     return vsx_LoadParameterSetOnDevice_ptr(vsx);
02352 }
02353
02354 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.SaveParameterSetOnDevice
02355 static VsxStatusCode (DNNE_CALLTYPE* vsx_SaveParameterSetOnDevice_ptr) (VsxSystemHandle* vsx);
02356 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SaveParameterSetOnDevice (VsxSystemHandle* vsx)
02357 {
02358     if (vsx_SaveParameterSetOnDevice_ptr == NULL)
02359     {
02360         const char_t* methodName = DNNE_STR("SaveParameterSetOnDevice");
02361         vsx_SaveParameterSetOnDevice_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle*
vsx))get_fast_callable_managed_function(t5_name, methodName);
02362     }
02363     return vsx_SaveParameterSetOnDevice_ptr(vsx);
02364 }
02365
02366 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.UploadParameterSet
02367 static VsxStatusCode (DNNE_CALLTYPE* vsx_UploadParameterSet_ptr) (VsxSystemHandle* vsx, const char*
fileName);
02368 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterSet (VsxSystemHandle* vsx, const
char* fileName)
02369 {
02370     if (vsx_UploadParameterSet_ptr == NULL)
02371     {
02372         const char_t* methodName = DNNE_STR("UploadParameterSet");
02373         vsx_UploadParameterSet_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx, const char*
fileName))get_fast_callable_managed_function(t5_name, methodName);
02374     }
02375     return vsx_UploadParameterSet_ptr(vsx, fileName);
02376 }
02377
02378 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameter.DownloadParameterSet
02379 static VsxStatusCode (DNNE_CALLTYPE* vsx_DownloadParameterSet_ptr) (VsxSystemHandle* vsx, const char*
fileName);
02380 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_DownloadParameterSet (VsxSystemHandle* vsx,
const char* fileName)
02381 {
02382     if (vsx_DownloadParameterSet_ptr == NULL)
02383     {
02384         const char_t* methodName = DNNE_STR("DownloadParameterSet");
02385         vsx_DownloadParameterSet_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx, const
char* fileName))get_fast_callable_managed_function(t5_name, methodName);
02386     }
02387     return vsx_DownloadParameterSet_ptr(vsx, fileName);
02388 }
02389
02390 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetParameterList
02391 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetParameterList_ptr) (VsxSystemHandle* vsx,
VsxParameterList** parameterListData);
02392 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetParameterList (VsxSystemHandle* vsx,
VsxParameterList** parameterListData)
02393 {
02394     if (vsx_GetParameterList_ptr == NULL)
02395     {
02396         const char_t* methodName = DNNE_STR("GetParameterList");
02397         vsx_GetParameterList_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx,
VsxParameterList** parameterListData))get_fast_callable_managed_function(t6_name, methodName);
02398     }
02399     return vsx_GetParameterList_ptr(vsx, parameterListData);
02400 }
02401
02402 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.UploadParameterList
02403 static VsxStatusCode (DNNE_CALLTYPE* vsx_UploadParameterList_ptr) (VsxSystemHandle* vsx,
VsxParameterList* parameterListData);
02404 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_UploadParameterList (VsxSystemHandle* vsx,
VsxParameterList* parameterListData)
02405 {
02406     if (vsx_UploadParameterList_ptr == NULL)
02407     {
02408         const char_t* methodName = DNNE_STR("UploadParameterList");
02409         vsx_UploadParameterList_ptr = (VsxStatusCode (DNNE_CALLTYPE*) (VsxSystemHandle* vsx,
VsxParameterList* parameterListData))get_fast_callable_managed_function(t6_name, methodName);
02410     }
02411     return vsx_UploadParameterList_ptr(vsx, parameterListData);
02412 }
02413

```

```

02414 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterString
02415 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterString_ptr) (VsxSystemHandle* vsx, const
02416 VsxParameter* parameter, const char* value);
02416 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterString (VsxSystemHandle* vsx,
02417 const VsxParameter* parameter, const char* value)
02417 {
02418     if (vsx_SetSingleParameterString_ptr == NULL)
02419     {
02420         const char_t* methodName = DNNE_STR("SetSingleParameterString");
02421         vsx_SetSingleParameterString_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const
02422 VsxParameter* parameter, const char* value) get_fast_callable_managed_function(t6_name, methodName);
02422     }
02423     return vsx_SetSingleParameterString_ptr(vsx, parameter, value);
02424 }
02425
02426 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterDouble
02427 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterDouble_ptr) (VsxSystemHandle* vsx, const
02428 VsxParameter* parameter, double value);
02428 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterDouble (VsxSystemHandle* vsx,
02429 const VsxParameter* parameter, double value)
02429 {
02430     if (vsx_SetSingleParameterDouble_ptr == NULL)
02431     {
02432         const char_t* methodName = DNNE_STR("SetSingleParameterDouble");
02433         vsx_SetSingleParameterDouble_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const
02434 VsxParameter* parameter, double value) get_fast_callable_managed_function(t6_name, methodName);
02434     }
02435     return vsx_SetSingleParameterDouble_ptr(vsx, parameter, value);
02436 }
02437
02438 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.SetSingleParameterInt32
02439 static VsxStatusCode (DNNE_CALLTYPE* vsx_SetSingleParameterInt32_ptr) (VsxSystemHandle* vsx, const
02440 VsxParameter* parameter, int32_t value);
02440 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_SetSingleParameterInt32 (VsxSystemHandle* vsx,
02441 const VsxParameter* parameter, int32_t value)
02441 {
02442     if (vsx_SetSingleParameterInt32_ptr == NULL)
02443     {
02444         const char_t* methodName = DNNE_STR("SetSingleParameterInt32");
02445         vsx_SetSingleParameterInt32_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const
02446 VsxParameter* parameter, int32_t value) get_fast_callable_managed_function(t6_name, methodName);
02446     }
02447     return vsx_SetSingleParameterInt32_ptr(vsx, parameter, value);
02448 }
02449
02450 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.GetSingleParameter
02451 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetSingleParameter_ptr) (VsxSystemHandle* vsx, const
02452 VsxParameter* parameterIn, const VsxParameter** parameterOut);
02452 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetSingleParameter (VsxSystemHandle* vsx, const
02453 VsxParameter* parameterIn, const VsxParameter** parameterOut)
02453 {
02454     if (vsx_GetSingleParameter_ptr == NULL)
02455     {
02456         const char_t* methodName = DNNE_STR("GetSingleParameter");
02457         vsx_GetSingleParameter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxSystemHandle* vsx, const
02458 VsxParameter* parameterIn, const VsxParameter**
02459 parameterOut) get_fast_callable_managed_function(t6_name, methodName);
02458     }
02459     return vsx_GetSingleParameter_ptr(vsx, parameterIn, parameterOut);
02460 }
02461
02462 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameter
02463 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseParameter_ptr) (const VsxParameter** pParameter);
02464 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameter (const VsxParameter**
02465 pParameter)
02465 {
02466     if (vsx_ReleaseParameter_ptr == NULL)
02467     {
02468         const char_t* methodName = DNNE_STR("ReleaseParameter");
02469         vsx_ReleaseParameter_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (const VsxParameter**
02470 pParameter) get_fast_callable_managed_function(t6_name, methodName);
02470     }
02471     return vsx_ReleaseParameter_ptr(pParameter);
02472 }
02473
02474 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsParameterList.ReleaseParameterList
02475 static VsxStatusCode (DNNE_CALLTYPE* vsx_ReleaseParameterList_ptr) (VsxParameterList** pParameterList);
02476 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_ReleaseParameterList (VsxParameterList**
02477 pParameterList)
02477 {
02478     if (vsx_ReleaseParameterList_ptr == NULL)
02479     {
02480         const char_t* methodName = DNNE_STR("ReleaseParameterList");
02481         vsx_ReleaseParameterList_ptr = (VsxStatusCode(DNNE_CALLTYPE*)) (VsxParameterList**
02482 pParameterList) get_fast_callable_managed_function(t6_name, methodName);
02482     }
02483     return vsx_ReleaseParameterList_ptr(pParameterList);

```



```

02484 }
02485
02486 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultXml
02487 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultXml_ptr) (VsxDataContainerHandle* dch, const char*
02488 resultId, const char** result);
02488 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultXml(VsxDataContainerHandle* dch, const
02489 char* resultId, const char** result)
02489 {
02490     if (vsx_GetResultXml_ptr == NULL)
02491     {
02492         const char_t* methodName = DNNE_STR("GetResultXml");
02493         vsx_GetResultXml_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch, const char*
02494 resultId, const char** result))get_fast_callable_managed_function(t7_name, methodName);
02494     }
02495     return vsx_GetResultXml_ptr(dch, resultId, result);
02496 }
02497
02498 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementString
02499 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementString_ptr) (VsxDataContainerHandle* dch,
02500 const char* resultId, const char* xPath, const char** result);
02500 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementString(VsxDataContainerHandle*
02501 dch, const char* resultId, const char* xPath, const char** result)
02501 {
02502     if (vsx_GetResultElementString_ptr == NULL)
02503     {
02504         const char_t* methodName = DNNE_STR("GetResultElementString");
02505         vsx_GetResultElementString_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch,
02506 const char* resultId, const char* xPath, const char**
02507 result))get_fast_callable_managed_function(t7_name, methodName);
02506     }
02507     return vsx_GetResultElementString_ptr(dch, resultId, xPath, result);
02508 }
02509
02510 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt32
02511 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementInt32_ptr) (VsxDataContainerHandle* dch, const
02512 char* resultId, const char* xPath, int32_t* result);
02512 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt32(VsxDataContainerHandle*
02513 dch, const char* resultId, const char* xPath, int32_t* result)
02513 {
02514     if (vsx_GetResultElementInt32_ptr == NULL)
02515     {
02516         const char_t* methodName = DNNE_STR("GetResultElementInt32");
02517         vsx_GetResultElementInt32_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch,
02518 const char* resultId, const char* xPath, int32_t* result))get_fast_callable_managed_function(t7_name,
02519 methodName);
02518     }
02519     return vsx_GetResultElementInt32_ptr(dch, resultId, xPath, result);
02520 }
02521
02522 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementInt64
02523 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementInt64_ptr) (VsxDataContainerHandle* dch, const
02524 char* resultId, const char* xPath, LOCAL_INT64_T* result);
02524 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementInt64(VsxDataContainerHandle*
02525 dch, const char* resultId, const char* xPath, LOCAL_INT64_T* result)
02525 {
02526     if (vsx_GetResultElementInt64_ptr == NULL)
02527     {
02528         const char_t* methodName = DNNE_STR("GetResultElementInt64");
02529         vsx_GetResultElementInt64_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch,
02530 const char* resultId, const char* xPath, LOCAL_INT64_T*
02531 result))get_fast_callable_managed_function(t7_name, methodName);
02530     }
02531     return vsx_GetResultElementInt64_ptr(dch, resultId, xPath, result);
02532 }
02533
02534 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsResult.GetResultElementDouble
02535 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetResultElementDouble_ptr) (VsxDataContainerHandle* dch,
02536 const char* resultId, const char* xPath, double* result);
02536 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetResultElementDouble(VsxDataContainerHandle*
02537 dch, const char* resultId, const char* xPath, double* result)
02537 {
02538     if (vsx_GetResultElementDouble_ptr == NULL)
02539     {
02540         const char_t* methodName = DNNE_STR("GetResultElementDouble");
02541         vsx_GetResultElementDouble_ptr = (VsxStatusCode(DNNE_CALLTYPE*) (VsxDataContainerHandle* dch,
02542 const char* resultId, const char* xPath, double* result))get_fast_callable_managed_function(t7_name,
02543 methodName);
02542     }
02543     return vsx_GetResultElementDouble_ptr(dch, resultId, xPath, result);
02544 }
02545
02546 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.GetAllDeviceStatusData
02547 static VsxStatusCode (DNNE_CALLTYPE* vsx_GetAllDeviceStatusData_ptr) (VsxSystemHandle* vsx,
02548 VsxStatusItemList** statusItemListData);
02548 DNNE_EXTERN_C DNNE_API VsxStatusCode DNNE_CALLTYPE vsx_GetAllDeviceStatusData(VsxSystemHandle* vsx,
02549 VsxStatusItemList** statusItemListData)
02549 {

```

```

02550     if (vsx_GetAllDeviceStatusData_ptr == NULL)
02551     {
02552         const char_t* methodName = DNNE_STR("GetAllDeviceStatusData");
02553         vsx_GetAllDeviceStatusData_ptr = (VsxCStatusCode(DNNE_CALLTYPE*) (VsxCSystemHandle* vsx,
VsxCStatusItemList** statusItemListData))get_fast_callable_managed_function(t8_name, methodName);
02554     }
02555     return vsx_GetAllDeviceStatusData_ptr(vsx, statusItemListData);
02556 }
02557
02558 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.ReleaseStatusItemList
02559 static VsxCStatusCode (DNNE_CALLTYPE* vsx_ReleaseStatusItemList_ptr) (VsxCStatusItemList**
pStatusItemList);
02560 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_ReleaseStatusItemList(VsxCStatusItemList**
pStatusItemList)
02561 {
02562     if (vsx_ReleaseStatusItemList_ptr == NULL)
02563     {
02564         const char_t* methodName = DNNE_STR("ReleaseStatusItemList");
02565         vsx_ReleaseStatusItemList_ptr = (VsxCStatusCode(DNNE_CALLTYPE*) (VsxCStatusItemList**
pStatusItemList))get_fast_callable_managed_function(t8_name, methodName);
02566     }
02567     return vsx_ReleaseStatusItemList_ptr(pStatusItemList);
02568 }
02569
02570 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.RegisterOnDeviceStatusReceived
02571 static VsxCStatusCode (DNNE_CALLTYPE* vsx_RegisterOnDeviceStatusReceived_ptr) (VsxCSystemHandle* vsx,
vsxCOnDeviceStatusReceived fptr);
02572 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_RegisterOnDeviceStatusReceived(VsxCSystemHandle*
vsx, vsxCOnDeviceStatusReceived fptr)
02573 {
02574     if (vsx_RegisterOnDeviceStatusReceived_ptr == NULL)
02575     {
02576         const char_t* methodName = DNNE_STR("RegisterOnDeviceStatusReceived");
02577         vsx_RegisterOnDeviceStatusReceived_ptr = (VsxCStatusCode(DNNE_CALLTYPE*) (VsxCSystemHandle* vsx,
vsxCOnDeviceStatusReceived fptr))get_fast_callable_managed_function(t8_name, methodName);
02578     }
02579     return vsx_RegisterOnDeviceStatusReceived_ptr(vsx, fpfptr);
02580 }
02581
02582 // Computed from
PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.DeregisterOnDeviceStatusReceived
02583 static VsxCStatusCode (DNNE_CALLTYPE* vsx_DeregisterOnDeviceStatusReceived_ptr) (VsxCSystemHandle* vsx);
02584 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE
vsx_DeregisterOnDeviceStatusReceived(VsxCSystemHandle* vsx)
02585 {
02586     if (vsx_DeregisterOnDeviceStatusReceived_ptr == NULL)
02587     {
02588         const char_t* methodName = DNNE_STR("DeregisterOnDeviceStatusReceived");
02589         vsx_DeregisterOnDeviceStatusReceived_ptr = (VsxCStatusCode(DNNE_CALLTYPE*) (VsxCSystemHandle*
vsx))get_fast_callable_managed_function(t8_name, methodName);
02590     }
02591     return vsx_DeregisterOnDeviceStatusReceived_ptr(vsx);
02592 }
02593
02594 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.SubscribeToDeviceStatusData
02595 static VsxCStatusCode (DNNE_CALLTYPE* vsx_SubscribeToDeviceStatusData_ptr) (VsxCSystemHandle* vsx);
02596 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_SubscribeToDeviceStatusData(VsxCSystemHandle*
vsx)
02597 {
02598     if (vsx_SubscribeToDeviceStatusData_ptr == NULL)
02599     {
02600         const char_t* methodName = DNNE_STR("SubscribeToDeviceStatusData");
02601         vsx_SubscribeToDeviceStatusData_ptr = (VsxCStatusCode(DNNE_CALLTYPE*) (VsxCSystemHandle*
vsx))get_fast_callable_managed_function(t8_name, methodName);
02602     }
02603     return vsx_SubscribeToDeviceStatusData_ptr(vsx);
02604 }
02605
02606 // Computed from PF.VsxProtocolDriver.Wrapper.VsxExportsStatusItemList.UnsubscribeToDeviceStatusData
02607 static VsxCStatusCode (DNNE_CALLTYPE* vsx_UnsubscribeToDeviceStatusData_ptr) (VsxCSystemHandle* vsx);
02608 DNNE_EXTERN_C DNNE_API VsxCStatusCode DNNE_CALLTYPE vsx_UnsubscribeToDeviceStatusData(VsxCSystemHandle*
vsx)
02609 {
02610     if (vsx_UnsubscribeToDeviceStatusData_ptr == NULL)
02611     {
02612         const char_t* methodName = DNNE_STR("UnsubscribeToDeviceStatusData");
02613         vsx_UnsubscribeToDeviceStatusData_ptr = (VsxCStatusCode(DNNE_CALLTYPE*) (VsxCSystemHandle*
vsx))get_fast_callable_managed_function(t8_name, methodName);
02614     }
02615     return vsx_UnsubscribeToDeviceStatusData_ptr(vsx);
02616 }
02617
02618 #endif // DNNE_COMPILE_AS_SOURCE

```





## Index

- [\\_VsxCaptureInformation, 13](#)
  - [exposureTime, 14](#)
  - [frameCounter, 14](#)
  - [gain, 15](#)
  - [illumination, 15](#)
  - [jobId, 14](#)
  - [parameterId, 14](#)
  - [rotaryEncoder, 14](#)
  - [timestamp, 14](#)
  - [triggerCounter, 14](#)
  - [triggerSource, 15](#)
- [\\_VsxDataContainerHandle, 15](#)
  - [handle, 15](#)
- [\\_VsxDevice, 15](#)
  - [baudrate, 17](#)
  - [busy, 17](#)
  - [comPort, 17](#)
  - [deviceVsxVersionMajor, 17](#)
  - [deviceVsxVersionMinor, 17](#)
  - [firmwareVersion, 16](#)
  - [gateway, 16](#)
  - [headAddress, 17](#)
  - [ipAddress, 16](#)
  - [isLoginNeeded, 17](#)
  - [macAddress, 16](#)
  - [networkMask, 16](#)
  - [sensorName, 16](#)
  - [sensorType, 16](#)
- [\\_VsxDeviceList, 17](#)
  - [devices, 18](#)
  - [length, 18](#)
- [\\_VsxDisparityDescriptor2, 18](#)
  - [baseline, 19](#)
  - [focalLength, 18](#)
  - [offsetLeftRectifiedToDisparityU, 19](#)
  - [offsetLeftRectifiedToDisparityV, 19](#)
  - [principalPointU, 18](#)
  - [principalPointV, 18](#)
- [\\_VsxImage, 19](#)
  - [axisMax, 20](#)
  - [axisMin, 20](#)
  - [coordinateOffset, 20](#)
  - [coordinateScale, 20](#)
  - [format, 19](#)
  - [frameCounter, 20](#)
  - [height, 20](#)
  - [invalidDataValue, 20](#)
  - [linePitch, 20](#)
  - [rawdata, 19](#)
  - [width, 20](#)
- [\\_VsxLineCoordinate, 21](#)
  - [c, 21](#)
  - [i, 22](#)
  - [q, 22](#)
  - [x, 21](#)
  - [y, 21](#)
  - [z, 21](#)
- [\\_VsxLineData, 22](#)
  - [countLines, 23](#)
  - [format, 22](#)
  - [frameCounter, 23](#)
  - [lines, 22](#)
  - [maxX, 23](#)
  - [maxZ, 23](#)
  - [minX, 23](#)
  - [minZ, 23](#)
  - [width, 23](#)
- [\\_VsxOlr2CaptureInformation, 23](#)
  - [autoTriggerFrameRate, 26](#)
  - [currentPosition, 24](#)
  - [frameCounter, 24](#)
  - [ioState, 24](#)
  - [ImaExposureTime1, 24](#)
  - [ImaExposureTime2, 25](#)
  - [ImaRoiLengthX, 25](#)
  - [ImaRoiLengthZ, 25](#)
  - [ImaRoiOffsetX, 25](#)
  - [ImaRoiOffsetZ, 25](#)
  - [ImbExposureTime1, 25](#)
  - [ImbExposureTime2, 25](#)
  - [ImbRoiLengthX, 25](#)
  - [ImbRoiLengthZ, 26](#)
  - [ImbRoiOffsetX, 25](#)
  - [ImbRoiOffsetZ, 25](#)
  - [timestamp, 24](#)
  - [triggerCounter, 24](#)
  - [triggerSource, 26](#)
- [\\_VsxOlr2ModbusData, 26](#)
  - [activationTimer, 26](#)
  - [compareBuffer, 26](#)
  - [robotData, 27](#)
  - [targetPosition, 26](#)
- [\\_VsxParameter, 27](#)
  - [configId, 28](#)
  - [configVersion, 28](#)
  - [enumItemList, 28](#)
  - [enumItemListLength, 28](#)
  - [name, 28](#)
  - [parameterId, 28](#)
  - [settingsVersion, 28](#)
  - [valueDouble, 27](#)
  - [valueInt, 27](#)
  - [valueString, 27](#)
  - [valueType, 27](#)
- [\\_VsxParameterEnumItem, 28](#)
  - [id, 29](#)
  - [name, 29](#)
- [\\_VsxParameterList, 29](#)
  - [length, 29](#)
  - [parameters, 29](#)

- `_VsxStatusItem`, 29
  - `configurationClass`, 30
  - `configVersion`, 31
  - `name`, 30
  - `sensorTime`, 31
  - `settingsVersion`, 31
  - `statusItemId`, 30
  - `time`, 31
  - `valueDouble`, 30
  - `valueInt`, 30
  - `valueString`, 30
  - `valueType`, 30
- `_VsxStatusItemList`, 31
  - `length`, 31
  - `statusItems`, 31
- `_VsxSystemHandle`, 32
  - `handle`, 32
- `_VsxTagList`, 32
  - `length`, 33
  - `tags`, 33
- `_VsxTransformation`, 33
  - `quaternionQ0`, 33
  - `quaternionQ1`, 34
  - `quaternionQ2`, 34
  - `quaternionQ3`, 34
  - `translationTX`, 33
  - `translationTY`, 33
  - `translationTZ`, 33
- `_vsxDeviceStatusScope`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 60
- `_vsxDisconnectEvent`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 57
- `_vsxImageData2Format`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 58
- `_vsxLineDataFormat`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 59
- `_vsxParameterValueType`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 59
- `_vsxSerialConnectionType`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 57
- `_vsxSessionTypes`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 58
- `_vsxStatusCode`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 54
- `_vsxStatusItemValueType`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 59
- `_vsxStrategy`
  - `PF.VsxProtocolDriver.WrapperNE.h`, 53
- `01_Introduction.md`, 34
- `02_Usage_with_c.md`, 34
- `03_Examples.md`, 34
- `04_Device_parameter.md`, 34
- `05_Changelog.md`, 34
- `activationTimer`
  - `_VsxOlr2ModbusData`, 26
- `autoTriggerFrameRate`
  - `_VsxOlr2CaptureInformation`, 26
- `axisMax`
  - `_VsxImage`, 20
- `axisMin`
  - `_VsxImage`, 20
- `baseline`
  - `_VsxDisparityDescriptor2`, 19
- `baudrate`
  - `_VsxDevice`, 17
- `busy`
  - `_VsxDevice`, 17
- `c`
  - `_VsxLineCoordinate`, 21
- `Changelog`, 10
- `compareBuffer`
  - `_VsxOlr2ModbusData`, 26
- `comPort`
  - `_VsxDevice`, 17
- `configId`
  - `_VsxParameter`, 28
- `configurationClass`
  - `_VsxStatusItem`, 30
- `configVersion`
  - `_VsxParameter`, 28
  - `_VsxStatusItem`, 31
- `coordinateOffset`
  - `_VsxImage`, 20
- `coordinateScale`
  - `_VsxImage`, 20
- `countLines`
  - `_VsxLineData`, 23
- `currentPosition`
  - `_VsxOlr2CaptureInformation`, 24
- `Device parameter`, 9
- `devices`
  - `_VsxDeviceList`, 18
- `deviceVsxVersionMajor`
  - `_VsxDevice`, 17
- `deviceVsxVersionMinor`
  - `_VsxDevice`, 17
- `dnne.h`, 34, 37
  - `dnne_abort`, 37
  - `DNNE_API`, 35
  - `DNNE_CALLTYPE`, 35
  - `DNNE_CALLTYPE_CDECL`, 35
  - `DNNE_CALLTYPE_FASTCALL`, 35
  - `DNNE_CALLTYPE_STDCALL`, 35
  - `DNNE_CALLTYPE_THISCALL`, 35
  - `DNNE_EXTERN_C`, 36
  - `DNNE_LINUX`, 35
  - `DNNE_STR`, 36
  - `DNNE_SUCCESS`, 36
  - `DNNE_WCHAR`, 36
  - `failure_fn`, 36
  - `failure_load_export`, 36
  - `failure_load_runtime`, 36
  - `failure_type`, 36
  - `preload_runtime`, 37

- set\_failure\_callback, 36
  - try\_preload\_runtime, 37
- dnne\_abort
  - dnne.h, 37
- DNNE\_API
  - dnne.h, 35
- DNNE\_CALLTYPE
  - dnne.h, 35
- DNNE\_CALLTYPE\_CDECL
  - dnne.h, 35
- DNNE\_CALLTYPE\_FASTCALL
  - dnne.h, 35
- DNNE\_CALLTYPE\_STDCALL
  - dnne.h, 35
- DNNE\_CALLTYPE\_THISCALL
  - dnne.h, 35
- DNNE\_EXTERN\_C
  - dnne.h, 36
- DNNE\_LINUX
  - dnne.h, 35
- DNNE\_STR
  - dnne.h, 36
- DNNE\_SUCCESS
  - dnne.h, 36
- DNNE\_WCHAR
  - dnne.h, 36
- enumItemList
  - \_VsxDParameter, 28
- enumItemListLength
  - \_VsxDParameter, 28
- Examples, 8
- exposureTime
  - \_VsxCaptureInformation, 14
- failure\_fn
  - dnne.h, 36
- failure\_load\_export
  - dnne.h, 36
- failure\_load\_runtime
  - dnne.h, 36
- failure\_type
  - dnne.h, 36
- firmwareVersion
  - \_VsxDDevice, 16
- focalLength
  - \_VsxDParityDescriptor2, 18
- format
  - \_VsxDImage, 19
  - \_VsxDLineData, 22
- frameCounter
  - \_VsxCaptureInformation, 14
  - \_VsxDImage, 20
  - \_VsxDLineData, 23
  - \_VsxDI2r2CaptureInformation, 24
- gain
  - \_VsxCaptureInformation, 15
- gateway
  - \_VsxDDevice, 16
- handle
  - \_VsxDDataContainerHandle, 15
  - \_VsxDSystemHandle, 32
- headAddress
  - \_VsxDDevice, 17
- height
  - \_VsxDImage, 20
- i
  - \_VsxDLineCoordinate, 22
- id
  - \_VsxDParameterEnumItem, 29
- illumination
  - \_VsxCaptureInformation, 15
- Introduction, 1
- invalidDataValue
  - \_VsxDImage, 20
- ioState
  - \_VsxDI2r2CaptureInformation, 24
- ipAddress
  - \_VsxDDevice, 16
- isLoginNeeded
  - \_VsxDDevice, 17
- jobId
  - \_VsxCaptureInformation, 14
- length
  - \_VsxDDeviceList, 18
  - \_VsxDParameterList, 29
  - \_VsxDStatusItemList, 31
  - \_VsxDTagList, 33
- linePitch
  - \_VsxDImage, 20
- lines
  - \_VsxDLineData, 22
- ImaExposureTime1
  - \_VsxDI2r2CaptureInformation, 24
- ImaExposureTime2
  - \_VsxDI2r2CaptureInformation, 25
- ImaRoiLengthX
  - \_VsxDI2r2CaptureInformation, 25
- ImaRoiLengthZ
  - \_VsxDI2r2CaptureInformation, 25
- ImaRoiOffsetX
  - \_VsxDI2r2CaptureInformation, 25
- ImaRoiOffsetZ
  - \_VsxDI2r2CaptureInformation, 25
- ImbExposureTime1
  - \_VsxDI2r2CaptureInformation, 25
- ImbExposureTime2
  - \_VsxDI2r2CaptureInformation, 25
- ImbRoiLengthX
  - \_VsxDI2r2CaptureInformation, 25
- ImbRoiLengthZ
  - \_VsxDI2r2CaptureInformation, 26
- ImbRoiOffsetX

- [\\_VsxOlr2CaptureInformation, 25](#)
  - [ImbRoiOffsetZ](#)
    - [\\_VsxOlr2CaptureInformation, 25](#)
  - [LOCAL\\_INT64\\_T](#)
    - [PF.VsxProtocolDriver.WrapperNE.h, 49](#)
  - [LOCAL\\_UINT64\\_T](#)
    - [PF.VsxProtocolDriver.WrapperNE.h, 49](#)
- [macAddress](#)
  - [\\_VsxDevice, 16](#)
- [maxX](#)
  - [\\_VsxLineData, 23](#)
- [maxZ](#)
  - [\\_VsxLineData, 23](#)
- [minX](#)
  - [\\_VsxLineData, 23](#)
- [minZ](#)
  - [\\_VsxLineData, 23](#)
- [name](#)
  - [\\_VsxParameter, 28](#)
  - [\\_VsxParameterEnumItem, 29](#)
  - [\\_VsxStatusItem, 30](#)
- [networkMask](#)
  - [\\_VsxDevice, 16](#)
- [offsetLeftRectifiedToDisparityU](#)
  - [\\_VsxDisparityDescriptor2, 19](#)
- [offsetLeftRectifiedToDisparityV](#)
  - [\\_VsxDisparityDescriptor2, 19](#)
- [parameterId](#)
  - [\\_VsxCaptureInformation, 14](#)
  - [\\_VsxParameter, 28](#)
- [parameters](#)
  - [\\_VsxParameterList, 29](#)
- [PF.VsxProtocolDriver.WrapperNE.h, 38, 99](#)
  - [\\_vsxDeviceStatusScope, 60](#)
  - [\\_vsxDisconnectEvent, 57](#)
  - [\\_vsxImageData2Format, 58](#)
  - [\\_vsxLineDataFormat, 59](#)
  - [\\_vsxParameterValueType, 59](#)
  - [\\_vsxSerialConnectionType, 57](#)
  - [\\_vsxSessionTypes, 58](#)
  - [\\_vsxStatusCode, 54](#)
  - [\\_vsxStatusItemValueType, 59](#)
  - [\\_vsxStrategy, 53](#)
  - [LOCAL\\_INT64\\_T, 49](#)
  - [LOCAL\\_UINT64\\_T, 49](#)
  - [vsx\\_Connect, 65](#)
  - [vsx\\_ConnectAndLogin, 65](#)
  - [vsx\\_ConnectEx, 65](#)
  - [vsx\\_ConnectExAndLogin, 66](#)
  - [vsx\\_DeregisterOnDeviceStatusReceived, 98](#)
  - [vsx\\_DeregisterOnDisconnect, 68](#)
  - [vsx\\_DeregisterOnSessionMessageReceived, 69](#)
  - [VSX\\_DEVICE\\_STATUS\\_SCOPE\\_FULL, 60](#)
  - [VSX\\_DEVICE\\_STATUS\\_SCOPE\\_MULTI, 60](#)
  - [vsx\\_Disconnect, 67](#)

- [VSX\\_DISCONNECT\\_EVENT\\_CONNECTION\\_ERROR, 58](#)
- [VSX\\_DISCONNECT\\_EVENT\\_DISCONNECT\\_CALLED, 58](#)
- [VSX\\_DISCONNECT\\_EVENT\\_REMOTE\\_HOST\\_CONNECTION\\_CLOSED, 58](#)
- [vsx\\_DownloadParameterSet, 90](#)
- [vsx\\_GetAllDeviceStatusData, 97](#)
- [vsx\\_GetCachedContainer, 74](#)
- [vsx\\_GetCaptureInformation, 78](#)
- [vsx\\_GetConnected, 67](#)
- [vsx\\_GetDataContainer, 73](#)
- [vsx\\_GetDeviceInformation, 82](#)
- [vsx\\_GetDisparityDescriptor2, 77](#)
- [vsx\\_GetDynamicContainerQueueSize, 81](#)
- [vsx\\_GetErrorText, 61](#)
- [vsx\\_GetImage, 75](#)
- [vsx\\_GetLibraryVersion, 60](#)
- [vsx\\_GetLine, 76](#)
- [vsx\\_GetLogMessage, 84](#)
- [vsx\\_GetLogMessageQueueSize, 84](#)
- [vsx\\_GetMissingContainerFramesCounter, 81](#)
- [vsx\\_GetMissingLogMessagesCounter, 86](#)
- [vsx\\_GetNumberOfCachedContainers, 82](#)
- [vsx\\_GetOlr2CaptureInformation, 79](#)
- [vsx\\_GetOlr2ModbusData, 80](#)
- [vsx\\_GetParameterList, 92](#)
- [vsx\\_GetResultElementDouble, 96](#)
- [vsx\\_GetResultElementInt32, 95](#)
- [vsx\\_GetResultElementInt64, 96](#)
- [vsx\\_GetResultElementString, 95](#)
- [vsx\\_GetResultXml, 95](#)
- [vsx\\_GetSingleParameter, 94](#)
- [vsx\\_GetSingleParameterValue, 88](#)
- [vsx\\_GetSingleParameterValueDouble, 88](#)
- [vsx\\_GetSingleParameterValueInt32, 89](#)
- [vsx\\_GetTagList, 80](#)
- [vsx\\_GetTransformation, 78](#)
- [vsx\\_GetUdpDeviceList, 83](#)
- [vsx\\_GetWaitTimeout, 70](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_CONFIDENCE8, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_COORD3D\\_A16, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_COORD3D\\_A32F, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_COORD3D\\_B16, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_COORD3D\\_B32F, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_COORD3D\\_C16, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_COORD3D\\_C32F, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_MONO12, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_MONO16, 58](#)
- [VSX\\_IMAGE\\_DATA2\\_FORMAT\\_MONO8, 58](#)
- [vsx\\_InitSerialSensor, 62](#)

- [vsx\\_InitTcpSensor](#), 61
- [VSX\\_LINE\\_DATA\\_FORMAT\\_32BIT\\_MODE](#), 59
- [VSX\\_LINE\\_DATA\\_FORMAT\\_C](#), 59
- [VSX\\_LINE\\_DATA\\_FORMAT\\_I](#), 59
- [VSX\\_LINE\\_DATA\\_FORMAT\\_Q](#), 59
- [VSX\\_LINE\\_DATA\\_FORMAT\\_X](#), 59
- [VSX\\_LINE\\_DATA\\_FORMAT\\_Y](#), 59
- [VSX\\_LINE\\_DATA\\_FORMAT\\_Z](#), 59
- [vsx\\_LoadDefaultParameterSetOnDevice](#), 89
- [vsx\\_LoadParameterSetOnDevice](#), 89
- [vsx\\_Login](#), 66
- [vsx\\_Logout](#), 66
- [vsx\\_OnDeviceStatusReceived](#), 53
- [vsx\\_OnDisconnect](#), 50
- [vsx\\_OnSessionMessageReceived](#), 50
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_BASE64](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_BOOL](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_DOUBLE](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_ENUM](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_FLOAT](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_HEXSTRING](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_INT](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_INT16](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_IP](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_LONG](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_POINT](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_QUAD](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_RECTANGLE](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_STRING](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_UINT](#), 59
- [VSX\\_PARAMETER\\_VALUE\\_TYPE\\_UNKNOWN](#), 59
- [vsx\\_ReConnectAndLoginTcpDevice](#), 64
- [vsx\\_ReConnectSerialDevice](#), 64
- [vsx\\_ReConnectTcpDevice](#), 62
- [vsx\\_RegisterOnDeviceStatusReceived](#), 97
- [vsx\\_RegisterOnDisconnect](#), 68
- [vsx\\_RegisterOnSessionMessageReceived](#), 68
- [vsx\\_ReleaseCaptureInformation](#), 79
- [vsx\\_ReleaseDataContainer](#), 74
- [vsx\\_ReleaseDevice](#), 83
- [vsx\\_ReleaseDeviceList](#), 83
- [vsx\\_ReleaseDisparityDescriptor2](#), 77
- [vsx\\_ReleaseImage](#), 76
- [vsx\\_ReleaseLine](#), 76
- [vsx\\_ReleaseOlr2CaptureInformation](#), 79
- [vsx\\_ReleaseOlr2ModbusData](#), 80
- [vsx\\_ReleaseParameter](#), 94
- [vsx\\_ReleaseParameterList](#), 94
- [vsx\\_ReleaseSensor](#), 62
- [vsx\\_ReleaseStatusItemList](#), 97
- [vsx\\_ReleaseString](#), 60
- [vsx\\_ReleaseTagList](#), 81
- [vsx\\_ReleaseTransformation](#), 78
- [vsx\\_ResetDynamicContainerGrabber](#), 73
- [vsx\\_ResetLogMessageGrabber](#), 84
- [vsx\\_Save3DPointCloudData](#), 75
- [vsx\\_SaveData](#), 74
- [vsx\\_SaveParameterSetOnDevice](#), 90
- [vsx\\_SendFirmware](#), 71
- [vsx\\_SendSessionKeepAlive](#), 69
- [vsx\\_SendXmlDataMessage](#), 72
- [VSX\\_SERIAL\\_CONNECTION\\_TYPE\\_CANOPEN](#), 57
- [VSX\\_SERIAL\\_CONNECTION\\_TYPE\\_ETHERNET\\_IP](#), 57
- [VSX\\_SERIAL\\_CONNECTION\\_TYPE\\_PROFIBUS](#), 57
- [VSX\\_SERIAL\\_CONNECTION\\_TYPE\\_PROFINET](#), 57
- [VSX\\_SERIAL\\_CONNECTION\\_TYPE\\_RS485](#), 57
- [VSX\\_SERIAL\\_CONNECTION\\_TYPE\\_USB\\_SSI](#), 57
- [VSX\\_SESSION\\_TYPES\\_INITIAL\\_PASSWORD\\_REQUIRED](#), 58
- [VSX\\_SESSION\\_TYPES\\_LOGIN](#), 58
- [VSX\\_SESSION\\_TYPES\\_LOGIN\\_REPLY](#), 58
- [VSX\\_SESSION\\_TYPES\\_LOGIN\\_REQUIRED](#), 58
- [VSX\\_SESSION\\_TYPES\\_LOGOUT](#), 58
- [VSX\\_SESSION\\_TYPES\\_LOGOUT\\_REPLY](#), 58
- [VSX\\_SESSION\\_TYPES\\_SET\\_PASSWORD](#), 58
- [VSX\\_SESSION\\_TYPES\\_SET\\_PASSWORD\\_REPLY](#), 58
- [VSX\\_SESSION\\_TYPES\\_TIMEOUT](#), 58
- [VSX\\_SESSION\\_TYPES\\_TIMEOUT\\_ANNOUNCEMENT](#), 58
- [VSX\\_SESSION\\_TYPES\\_UNKNOWN](#), 58
- [vsx\\_SetNetworkSettings](#), 72
- [vsx\\_SetNetworkSettingsViaUdp](#), 73
- [vsx\\_SetPassword](#), 67
- [vsx\\_SetSingleParameterDouble](#), 93
- [vsx\\_SetSingleParameterInt32](#), 93
- [vsx\\_SetSingleParameterString](#), 92
- [vsx\\_SetSingleParameterValue](#), 86
- [vsx\\_SetSingleParameterValueDouble](#), 87
- [vsx\\_SetSingleParameterValueInt32](#), 87
- [vsx\\_SetWaitTimeout](#), 71
- [VSX\\_STATUS\\_ERROR\\_COMMAND\\_ZERO](#), 54
- [VSX\\_STATUS\\_ERROR\\_CONFIGURATION\\_ID\\_ZERO](#), 54
- [VSX\\_STATUS\\_ERROR\\_DATA\\_POINTER\\_CONTENTS\\_NOT\\_ZERO](#), 57
- [VSX\\_STATUS\\_ERROR\\_DATA\\_POINTER\\_CONTENTS\\_ZERO](#), 57
- [VSX\\_STATUS\\_ERROR\\_DATA\\_POINTER\\_ZERO](#), 57
- [VSX\\_STATUS\\_ERROR\\_DRIVER\\_CONNECTION](#), 54
- [VSX\\_STATUS\\_ERROR\\_DRIVER\\_DATA](#), 54
- [VSX\\_STATUS\\_ERROR\\_DRIVER\\_DEVICE](#), 54
- [VSX\\_STATUS\\_ERROR\\_DRIVER\\_GENERAL](#), 54
- [VSX\\_STATUS\\_ERROR\\_DRIVER\\_INIT](#), 54
- [VSX\\_STATUS\\_ERROR\\_DRIVER\\_INVALID\\_DATA](#), 54

VSX\_STATUS\_ERROR\_DRIVER\_LOAD\_FILE, 54  
 VSX\_STATUS\_ERROR\_DRIVER\_SAVE\_FILE, 54  
 VSX\_STATUS\_ERROR\_DRIVER\_TIMEOUT, 54  
 VSX\_STATUS\_ERROR\_ERROR\_TEXT\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_ERROR\_TEXT\_POINTER\_ZERO, 56  
 VSX\_STATUS\_ERROR\_EXCEPTION\_THROWN, 56  
 VSX\_STATUS\_ERROR\_FILENAME\_ZERO, 55  
 VSX\_STATUS\_ERROR\_GATEWAY\_ZERO, 56  
 VSX\_STATUS\_ERROR\_IMAGE\_TAG\_ZERO, 55  
 VSX\_STATUS\_ERROR\_INCORRECT\_MESSAGE\_FROM\_TAG, 57  
 VSX\_STATUS\_ERROR\_INPUT\_VALUE\_ZERO, 54  
 VSX\_STATUS\_ERROR\_INVALID\_DATA\_FORMAT, 56  
 VSX\_STATUS\_ERROR\_IP\_ADDRESS\_ZERO, 56  
 VSX\_STATUS\_ERROR\_LINE\_DATA\_TAG\_ZERO, 57  
 VSX\_STATUS\_ERROR\_LOG\_NOT\_ZERO, 55  
 VSX\_STATUS\_ERROR\_LOG\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_MAC\_ADDRESS\_ZERO, 56  
 VSX\_STATUS\_ERROR\_MISSING\_IP\_ADDRESS\_DECLARATION, 54  
 VSX\_STATUS\_ERROR\_MISSING\_LOGIN\_PASSWORD, 57  
 VSX\_STATUS\_ERROR\_MISSING\_LOGIN\_USERNAME, 57  
 VSX\_STATUS\_ERROR\_MISSING\_SERIALPORT\_DECLARATION, 54  
 VSX\_STATUS\_ERROR\_NETWORK\_MASK\_ZERO, 56  
 VSX\_STATUS\_ERROR\_NO\_ELEMENT\_FOUND, 56  
 VSX\_STATUS\_ERROR\_ON\_DISCONNECT\_CALLBACK\_ZERO, 56  
 VSX\_STATUS\_ERROR\_ON\_SESSION\_MESSAGE\_RECEIVE\_CALLBACK\_ZERO, 57  
 VSX\_STATUS\_ERROR\_OUTPUT\_VALUE\_NOT\_ZERO, 54  
 VSX\_STATUS\_ERROR\_OUTPUT\_VALUE\_POINTER\_ZERO, 54  
 VSX\_STATUS\_ERROR\_PARAMETER\_ID\_ZERO, 54  
 VSX\_STATUS\_ERROR\_POINT\_X\_ID\_ZERO, 55  
 VSX\_STATUS\_ERROR\_POINT\_Y\_ID\_ZERO, 55  
 VSX\_STATUS\_ERROR\_POINT\_Z\_ID\_ZERO, 55  
 VSX\_STATUS\_ERROR\_RESULT\_NOT\_ZERO, 55  
 VSX\_STATUS\_ERROR\_RESULT\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_RESULT\_TAG\_ZERO, 56  
 VSX\_STATUS\_ERROR\_SESSION, 54  
 VSX\_STATUS\_ERROR\_STRING, 54  
 VSX\_STATUS\_ERROR\_STRING\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_STRING\_ZERO, 55  
 VSX\_STATUS\_ERROR\_TAG\_ZERO, 56  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_ALLOCATE\_VSX\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_ALLOCATE\_VSX\_SYSTEM, 54  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_IMAGE\_ID\_IN\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_IMAGE\_TAG\_TO\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_LINE\_ID\_IN\_DATA\_CONTAINER, 57  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_LINE\_TAG\_TO\_DATA\_CONTAINER, 57  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_MESSAGE\_IN\_DATA\_CONTAINER, 57  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_X\_ID\_IN\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_X\_ID\_TO\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Y\_ID\_IN\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Y\_ID\_TO\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Z\_ID\_IN\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Z\_ID\_TO\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_RESULT\_ID\_IN\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_RESULT\_ID\_TO\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_TAG, 57  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_TAG\_IN\_DATA\_CONTAINER, 56  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_VSX\_DATA\_CONTAINER, 55  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_VSX\_SYSTEM, 55  
 VSX\_STATUS\_ERROR\_UNDEFINED\_CONNECTION\_TYPE\_VALUE, 56  
 VSX\_STATUS\_ERROR\_UNDEFINED\_STRATEGY\_VALUE, 56  
 VSX\_STATUS\_ERROR\_VALUE\_NOT\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VALUE\_POINTER\_ZERO, 54  
 VSX\_STATUS\_ERROR\_VALUE\_ZERO, 54  
 VSX\_STATUS\_ERROR\_VERSION, 54  
 VSX\_STATUS\_ERROR\_VERSION\_NOT\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VERSION\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VSX\_CACHED\_CONTAINER\_NOT\_FOUND, 56  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_NOT\_AVAILABLE, 55



VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_LIST\_POINTER\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_LIST\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_POINTER\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_IMAGE\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_IMAGE\_POINTER\_ZERO, 55  
 VSX\_STATUS\_ERROR\_VSX\_IMAGE\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_LINE\_DATA\_POINTER\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_LINE\_DATA\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_LINE\_NOT\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_IN\_POINTER\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_LIST\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_LIST\_POINTER\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_LIST\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_OUT\_POINTER\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_POINTER\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_ZERO, 57  
 VSX\_STATUS\_ERROR\_VSX\_STATUS\_ITEM\_LIST\_POINTER\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_STATUS\_ITEM\_LIST\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_STATUS\_ITEM\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_NOT\_AVAILABLE, 54  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_NOT\_ZERO, 54  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_POINTER\_ZERO, 54  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_ZERO, 54  
 VSX\_STATUS\_ERROR\_VSX\_TAG\_LIST\_NOT\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_TAG\_LIST\_POINTER\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_TAG\_LIST\_ZERO, 56  
 VSX\_STATUS\_ERROR\_VSX\_VALUE\_POINTER\_ZERO, 57  
 VSX\_STATUS\_ERROR\_XML\_COMMAND\_ZERO, 55  
 VSX\_STATUS\_ERROR\_XPATH\_ZERO, 56  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_BASE64, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_BOOL, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_DOUBLE, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_ENUM, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_FLOAT, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_HEXSTRING, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_INT, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_INT16, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_IP, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_LONG, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_POINT, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_QUAD, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_RECTANGLE, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_STRING, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_UINT, 60  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_UNKNOWN, 60  
 VSX\_STATUS\_SUCCESS, 54  
 VSX\_STRATEGY\_DROP\_OLDEST, 54  
 VSX\_STRATEGY\_DROP\_WRITE, 54  
 vsx\_SubscribeToDeviceStatusData, 98  
 vsx\_TestSystem, 69  
 vsx\_TestSystemEx, 70  
 vsx\_UnsubscribeToDeviceStatusData, 98  
 vsx\_UploadData, 71  
 vsx\_UploadParameterList, 92  
 vsx\_UploadParameterSet, 90  
 VsxCaptureInformation, 51  
 VsxDataContainerHandle, 50  
 VsxDevice, 52  
 VsxDeviceList, 52  
 VsxDeviceStatusScope, 53  
 VsxDisconnectEvent, 50  
 VsxDisparityDescriptor2, 51  
 VsxImage, 51  
 VsxImageData2Format, 51  
 VsxLineCoordinate, 51  
 VsxLineData, 51  
 VsxLineDataFormat, 51  
 VsxOlr2CaptureInformation, 52  
 VsxOlr2ModbusData, 52  
 VsxParameter, 52  
 VsxParameterEnumItem, 52  
 VsxParameterList, 53  
 VsxParameterValueType, 52  
 VsxSerialConnectionType, 50  
 VsxSessionTypes, 50



- VsxStatusCode, 49
- VsxStatusItem, 53
- VsxStatusItemList, 53
- VsxStatusItemValueType, 53
- VsxStrategy, 49
- VsxSystemHandle, 50
- VsxTagList, 52
- VsxTransformation, 51
- preload\_runtime
  - dnne.h, 37
- principalPointU
  - \_VsxDisparityDescriptor2, 18
- principalPointV
  - \_VsxDisparityDescriptor2, 18
- q
  - \_VsxLineCoordinate, 22
- quaternionQ0
  - \_VsxTransformation, 33
- quaternionQ1
  - \_VsxTransformation, 34
- quaternionQ2
  - \_VsxTransformation, 34
- quaternionQ3
  - \_VsxTransformation, 34
- rawdata
  - \_VsxImage, 19
- robotData
  - \_VsxOlr2ModbusData, 27
- rotaryEncoder
  - \_VsxCaptureInformation, 14
- sensorName
  - \_VsxDevice, 16
- sensorTime
  - \_VsxStatusItem, 31
- sensorType
  - \_VsxDevice, 16
- set\_failure\_callback
  - dnne.h, 36
- settingsVersion
  - \_VsxParameter, 28
  - \_VsxStatusItem, 31
- statusItemId
  - \_VsxStatusItem, 30
- statusItems
  - \_VsxStatusItemList, 31
- tags
  - \_VsxTagList, 33
- targetPosition
  - \_VsxOlr2ModbusData, 26
- time
  - \_VsxStatusItem, 31
- timestamp
  - \_VsxCaptureInformation, 14
  - \_VsxOlr2CaptureInformation, 24
- translationTX

- \_VsxTransformation, 33
- translationTY
  - \_VsxTransformation, 33
- translationTZ
  - \_VsxTransformation, 33
- triggerCounter
  - \_VsxCaptureInformation, 14
  - \_VsxOlr2CaptureInformation, 24
- triggerSource
  - \_VsxCaptureInformation, 15
  - \_VsxOlr2CaptureInformation, 26
- try\_preload\_runtime
  - dnne.h, 37
- Usage with C interface, 2
- valueDouble
  - \_VsxParameter, 27
  - \_VsxStatusItem, 30
- valueInt
  - \_VsxParameter, 27
  - \_VsxStatusItem, 30
- valueString
  - \_VsxParameter, 27
  - \_VsxStatusItem, 30
- valueType
  - \_VsxParameter, 27
  - \_VsxStatusItem, 30
- vsx\_Connect
  - PF.VsxProtocolDriver.WrapperNE.h, 65
- vsx\_ConnectAndLogin
  - PF.VsxProtocolDriver.WrapperNE.h, 65
- vsx\_ConnectEx
  - PF.VsxProtocolDriver.WrapperNE.h, 65
- vsx\_ConnectExAndLogin
  - PF.VsxProtocolDriver.WrapperNE.h, 66
- vsx\_DeregisterOnDeviceStatusReceived
  - PF.VsxProtocolDriver.WrapperNE.h, 98
- vsx\_DeregisterOnDisconnect
  - PF.VsxProtocolDriver.WrapperNE.h, 68
- vsx\_DeregisterOnSessionMessageReceived
  - PF.VsxProtocolDriver.WrapperNE.h, 69
- VSX\_DEVICE\_STATUS\_SCOPE\_FULL
  - PF.VsxProtocolDriver.WrapperNE.h, 60
- VSX\_DEVICE\_STATUS\_SCOPE\_MULTI
  - PF.VsxProtocolDriver.WrapperNE.h, 60
- vsx\_Disconnect
  - PF.VsxProtocolDriver.WrapperNE.h, 67
- VSX\_DISCONNECT\_EVENT\_CONNECTION\_ERROR
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_DISCONNECT\_EVENT\_DISCONNECT\_CALLED
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_DISCONNECT\_EVENT\_REMOTE\_HOST\_CONNECTION\_CLOSED
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- vsx\_DownloadParameterSet
  - PF.VsxProtocolDriver.WrapperNE.h, 90
- vsx\_GetAllDeviceStatusData
  - PF.VsxProtocolDriver.WrapperNE.h, 97
- vsx\_GetCachedContainer

- PF.VsxProtocolDriver.WrapperNE.h, 74
- vsx\_GetCaptureInformation
  - PF.VsxProtocolDriver.WrapperNE.h, 78
- vsx\_GetConnected
  - PF.VsxProtocolDriver.WrapperNE.h, 67
- vsx\_GetDataContainer
  - PF.VsxProtocolDriver.WrapperNE.h, 73
- vsx\_GetDeviceInformation
  - PF.VsxProtocolDriver.WrapperNE.h, 82
- vsx\_GetDisparityDescriptor2
  - PF.VsxProtocolDriver.WrapperNE.h, 77
- vsx\_GetDynamicContainerQueueSize
  - PF.VsxProtocolDriver.WrapperNE.h, 81
- vsx\_GetErrorText
  - PF.VsxProtocolDriver.WrapperNE.h, 61
- vsx\_GetImage
  - PF.VsxProtocolDriver.WrapperNE.h, 75
- vsx\_GetLibraryVersion
  - PF.VsxProtocolDriver.WrapperNE.h, 60
- vsx\_GetLine
  - PF.VsxProtocolDriver.WrapperNE.h, 76
- vsx\_GetLogMessage
  - PF.VsxProtocolDriver.WrapperNE.h, 84
- vsx\_GetLogMessageQueueSize
  - PF.VsxProtocolDriver.WrapperNE.h, 84
- vsx\_GetMissingContainerFramesCounter
  - PF.VsxProtocolDriver.WrapperNE.h, 81
- vsx\_GetMissingLogMessagesCounter
  - PF.VsxProtocolDriver.WrapperNE.h, 86
- vsx\_GetNumberOfCachedContainers
  - PF.VsxProtocolDriver.WrapperNE.h, 82
- vsx\_GetOlr2CaptureInformation
  - PF.VsxProtocolDriver.WrapperNE.h, 79
- vsx\_GetOlr2ModbusData
  - PF.VsxProtocolDriver.WrapperNE.h, 80
- vsx\_GetParameterList
  - PF.VsxProtocolDriver.WrapperNE.h, 92
- vsx\_GetResultElementDouble
  - PF.VsxProtocolDriver.WrapperNE.h, 96
- vsx\_GetResultElementInt32
  - PF.VsxProtocolDriver.WrapperNE.h, 95
- vsx\_GetResultElementInt64
  - PF.VsxProtocolDriver.WrapperNE.h, 96
- vsx\_GetResultElementString
  - PF.VsxProtocolDriver.WrapperNE.h, 95
- vsx\_GetResultXml
  - PF.VsxProtocolDriver.WrapperNE.h, 95
- vsx\_GetSingleParameter
  - PF.VsxProtocolDriver.WrapperNE.h, 94
- vsx\_GetSingleParameterValue
  - PF.VsxProtocolDriver.WrapperNE.h, 88
- vsx\_GetSingleParameterValueDouble
  - PF.VsxProtocolDriver.WrapperNE.h, 88
- vsx\_GetSingleParameterValueInt32
  - PF.VsxProtocolDriver.WrapperNE.h, 89
- vsx\_GetTagList
  - PF.VsxProtocolDriver.WrapperNE.h, 80
- vsx\_GetTransformation
  - PF.VsxProtocolDriver.WrapperNE.h, 78
- vsx\_GetUdpDeviceList
  - PF.VsxProtocolDriver.WrapperNE.h, 83
- vsx\_GetWaitTimeout
  - PF.VsxProtocolDriver.WrapperNE.h, 70
- VSX\_IMAGE\_DATA2\_FORMAT\_CONFIDENCE8
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_COORD3D\_A16
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_COORD3D\_A32F
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_COORD3D\_B16
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_COORD3D\_B32F
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_COORD3D\_C16
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_COORD3D\_C32F
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_MONO12
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_MONO16
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_IMAGE\_DATA2\_FORMAT\_MONO8
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- vsx\_InitSerialSensor
  - PF.VsxProtocolDriver.WrapperNE.h, 62
- vsx\_InitTcpSensor
  - PF.VsxProtocolDriver.WrapperNE.h, 61
- VSX\_LINE\_DATA\_FORMAT\_32BIT\_MODE
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- VSX\_LINE\_DATA\_FORMAT\_C
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- VSX\_LINE\_DATA\_FORMAT\_I
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- VSX\_LINE\_DATA\_FORMAT\_Q
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- VSX\_LINE\_DATA\_FORMAT\_X
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- VSX\_LINE\_DATA\_FORMAT\_Y
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- VSX\_LINE\_DATA\_FORMAT\_Z
  - PF.VsxProtocolDriver.WrapperNE.h, 59
- vsx\_LoadDefaultParameterSetOnDevice
  - PF.VsxProtocolDriver.WrapperNE.h, 89
- vsx\_LoadParameterSetOnDevice
  - PF.VsxProtocolDriver.WrapperNE.h, 89
- vsx\_Login
  - PF.VsxProtocolDriver.WrapperNE.h, 66
- vsx\_Logout
  - PF.VsxProtocolDriver.WrapperNE.h, 66
- vsx\_OnDeviceStatusReceived
  - PF.VsxProtocolDriver.WrapperNE.h, 53
- vsx\_OnDisconnect
  - PF.VsxProtocolDriver.WrapperNE.h, 50
- vsx\_OnSessionMessageReceived
  - PF.VsxProtocolDriver.WrapperNE.h, 50
- VSX\_PARAMETER\_VALUE\_TYPE\_BASE64

- PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_BOOL
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_DOUBLE
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_ENUM
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_FLOAT
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_HEXSTRING
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_INT
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_INT16
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_IP
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_LONG
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_POINT
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_QUAD
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_RECTANGLE
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_STRING
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_UINT
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- VSX\_PARAMETER\_VALUE\_TYPE\_UNKNOWN
  - PF.VsxProtocolDriver.WrapperNE.h, [59](#)
- vsx\_ReConnectAndLoginTcpDevice
  - PF.VsxProtocolDriver.WrapperNE.h, [64](#)
- vsx\_ReConnectSerialDevice
  - PF.VsxProtocolDriver.WrapperNE.h, [64](#)
- vsx\_ReConnectTcpDevice
  - PF.VsxProtocolDriver.WrapperNE.h, [62](#)
- vsx\_RegisterOnDeviceStatusReceived
  - PF.VsxProtocolDriver.WrapperNE.h, [97](#)
- vsx\_RegisterOnDisconnect
  - PF.VsxProtocolDriver.WrapperNE.h, [68](#)
- vsx\_RegisterOnSessionMessageReceived
  - PF.VsxProtocolDriver.WrapperNE.h, [68](#)
- vsx\_ReleaseCaptureInformation
  - PF.VsxProtocolDriver.WrapperNE.h, [79](#)
- vsx\_ReleaseDataContainer
  - PF.VsxProtocolDriver.WrapperNE.h, [74](#)
- vsx\_ReleaseDevice
  - PF.VsxProtocolDriver.WrapperNE.h, [83](#)
- vsx\_ReleaseDeviceList
  - PF.VsxProtocolDriver.WrapperNE.h, [83](#)
- vsx\_ReleaseDisparityDescriptor2
  - PF.VsxProtocolDriver.WrapperNE.h, [77](#)
- vsx\_ReleaseImage
  - PF.VsxProtocolDriver.WrapperNE.h, [76](#)
- vsx\_ReleaseLine
  - PF.VsxProtocolDriver.WrapperNE.h, [76](#)
- vsx\_ReleaseOlr2CaptureInformation
  - PF.VsxProtocolDriver.WrapperNE.h, [79](#)
- vsx\_ReleaseOlr2ModbusData
  - PF.VsxProtocolDriver.WrapperNE.h, [80](#)
- vsx\_ReleaseParameter
  - PF.VsxProtocolDriver.WrapperNE.h, [94](#)
- vsx\_ReleaseParameterList
  - PF.VsxProtocolDriver.WrapperNE.h, [94](#)
- vsx\_ReleaseSensor
  - PF.VsxProtocolDriver.WrapperNE.h, [62](#)
- vsx\_ReleaseStatusItemList
  - PF.VsxProtocolDriver.WrapperNE.h, [97](#)
- vsx\_ReleaseString
  - PF.VsxProtocolDriver.WrapperNE.h, [60](#)
- vsx\_ReleaseTagList
  - PF.VsxProtocolDriver.WrapperNE.h, [81](#)
- vsx\_ReleaseTransformation
  - PF.VsxProtocolDriver.WrapperNE.h, [78](#)
- vsx\_ResetDynamicContainerGrabber
  - PF.VsxProtocolDriver.WrapperNE.h, [73](#)
- vsx\_ResetLogMessageGrabber
  - PF.VsxProtocolDriver.WrapperNE.h, [84](#)
- vsx\_Save3DPointCloudData
  - PF.VsxProtocolDriver.WrapperNE.h, [75](#)
- vsx\_SaveData
  - PF.VsxProtocolDriver.WrapperNE.h, [74](#)
- vsx\_SaveParameterSetOnDevice
  - PF.VsxProtocolDriver.WrapperNE.h, [90](#)
- vsx\_SendFirmware
  - PF.VsxProtocolDriver.WrapperNE.h, [71](#)
- vsx\_SendSessionKeepAlive
  - PF.VsxProtocolDriver.WrapperNE.h, [69](#)
- vsx\_SendXmlDataMessage
  - PF.VsxProtocolDriver.WrapperNE.h, [72](#)
- VSX\_SERIAL\_CONNECTION\_TYPE\_CANOPEN
  - PF.VsxProtocolDriver.WrapperNE.h, [57](#)
- VSX\_SERIAL\_CONNECTION\_TYPE\_ETHERNET\_IP
  - PF.VsxProtocolDriver.WrapperNE.h, [57](#)
- VSX\_SERIAL\_CONNECTION\_TYPE\_PROFIBUS
  - PF.VsxProtocolDriver.WrapperNE.h, [57](#)
- VSX\_SERIAL\_CONNECTION\_TYPE\_PROFINET
  - PF.VsxProtocolDriver.WrapperNE.h, [57](#)
- VSX\_SERIAL\_CONNECTION\_TYPE\_RS485
  - PF.VsxProtocolDriver.WrapperNE.h, [57](#)
- VSX\_SERIAL\_CONNECTION\_TYPE\_USB\_SSI
  - PF.VsxProtocolDriver.WrapperNE.h, [57](#)
- VSX\_SESSION\_TYPES\_INITIAL\_PASSWORD\_REQUIRED
  - PF.VsxProtocolDriver.WrapperNE.h, [58](#)
- VSX\_SESSION\_TYPES\_LOGIN
  - PF.VsxProtocolDriver.WrapperNE.h, [58](#)
- VSX\_SESSION\_TYPES\_LOGIN\_REPLY
  - PF.VsxProtocolDriver.WrapperNE.h, [58](#)
- VSX\_SESSION\_TYPES\_LOGIN\_REQUIRED
  - PF.VsxProtocolDriver.WrapperNE.h, [58](#)
- VSX\_SESSION\_TYPES\_LOGOUT
  - PF.VsxProtocolDriver.WrapperNE.h, [58](#)
- VSX\_SESSION\_TYPES\_LOGOUT\_REPLY
  - PF.VsxProtocolDriver.WrapperNE.h, [58](#)
- VSX\_SESSION\_TYPES\_SET\_PASSWORD

- PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_SESSION\_TYPES\_SET\_PASSWORD\_REPLY
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_SESSION\_TYPES\_TIMEOUT
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_SESSION\_TYPES\_TIMEOUT\_ANNOUNCEMENT
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- VSX\_SESSION\_TYPES\_UNKNOWN
  - PF.VsxProtocolDriver.WrapperNE.h, 58
- vsx\_SetNetworkSettings
  - PF.VsxProtocolDriver.WrapperNE.h, 72
- vsx\_SetNetworkSettingsViaUdp
  - PF.VsxProtocolDriver.WrapperNE.h, 73
- vsx\_SetPassword
  - PF.VsxProtocolDriver.WrapperNE.h, 67
- vsx\_SetSingleParameterDouble
  - PF.VsxProtocolDriver.WrapperNE.h, 93
- vsx\_SetSingleParameterInt32
  - PF.VsxProtocolDriver.WrapperNE.h, 93
- vsx\_SetSingleParameterString
  - PF.VsxProtocolDriver.WrapperNE.h, 92
- vsx\_SetSingleParameterValue
  - PF.VsxProtocolDriver.WrapperNE.h, 86
- vsx\_SetSingleParameterValueDouble
  - PF.VsxProtocolDriver.WrapperNE.h, 87
- vsx\_SetSingleParameterValueInt32
  - PF.VsxProtocolDriver.WrapperNE.h, 87
- vsx\_SetWaitTimeout
  - PF.VsxProtocolDriver.WrapperNE.h, 71
- VSX\_STATUS\_ERROR\_COMMAND\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_CONFIGURATION\_ID\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DATA\_POINTER\_CONTENTS\_NOT\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_DATA\_POINTER\_CONTENTS\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_DATA\_POINTER\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_DRIVER\_CONNECTION
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_DATA
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_DEVICE
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_GENERAL
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_INIT
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_INVALID\_DATA
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_LOAD\_FILE
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_SAVE\_FILE
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_DRIVER\_TIMEOUT
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_ERROR\_TEXT\_NOT\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_ERROR\_TEXT\_POINTER\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_EXCEPTION\_THROWN
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_FILENAME\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_GATEWAY\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_IMAGE\_TAG\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_INCORRECT\_MESSAGE\_FROM\_TAG
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_INPUT\_VALUE\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_INVALID\_DATA\_FORMAT
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_IP\_ADDRESS\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_LINE\_DATA\_TAG\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_LOG\_NOT\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_LOG\_POINTER\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_MAC\_ADDRESS\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_MISSING\_IP\_ADDRESS\_DECLARATION
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_MISSING\_LOGIN\_PASSWORD
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_MISSING\_LOGIN\_USERNAME
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_MISSING\_SERIALPORT\_DECLARATION
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_NETWORK\_MASK\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_NO\_ELEMENT\_FOUND
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_ON\_DISCONNECT\_CALLBACK\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 56
- VSX\_STATUS\_ERROR\_ON\_SESSION\_MESSAGE\_RECEIVED\_CALLBACK
  - PF.VsxProtocolDriver.WrapperNE.h, 57
- VSX\_STATUS\_ERROR\_OUTPUT\_VALUE\_NOT\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_OUTPUT\_VALUE\_POINTER\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_PARAMETER\_ID\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 54
- VSX\_STATUS\_ERROR\_POINT\_X\_ID\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_POINT\_Y\_ID\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_POINT\_Z\_ID\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_RESULT\_NOT\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55
- VSX\_STATUS\_ERROR\_RESULT\_POINTER\_ZERO
  - PF.VsxProtocolDriver.WrapperNE.h, 55

PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_RESULT\_TAG\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_SESSION  
   PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_STRING  
   PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_STRING\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_STRING\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_TAG\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_ALLOCATE\_VSX\_DATA\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_ALLOCATE\_VSX\_DATA\_CONTAINER\_HANDLE  
   PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_IMAGE\_ID\_IN\_DATA\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_IMAGE\_TAG\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_LINE\_ID\_IN\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_LINE\_TAG\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_MESSAGE\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_X\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_X\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Y\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Y\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Z\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_POINT\_Z\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_RESULT\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_RESULT\_ID\_VSX\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_TAG  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_TAG\_IN\_DATA\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_VSX\_DATA\_CONTAINER  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNABLE\_TO\_FIND\_VSX\_SYSTEM  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_UNDEFINED\_CONNECTION\_TYPE\_VSX\_STATUS  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_UNDEFINED\_STRATEGY\_VALUE\_VSX\_STATUS  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VALUE\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VALUE\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VALUE\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VERSION  
   PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VERSION\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VERSION\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VSX\_CACHED\_CONTAINER\_NOT\_FOUND  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_NOT\_AVAILABLE  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VSX\_DATA\_CONTAINER\_HANDLE\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_LIST\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_LIST\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_DEVICE\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_IMAGE\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_IMAGE\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_VSX\_IMAGE\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_LINE\_DATA\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_LINE\_DATA\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_LINE\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_IN\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_LIST\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_LIST\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_LIST\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_NOT\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_OUT\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_POINTER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_PARAMETER\_ZERO  
   PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_VSX\_STATUS\_ITEM\_LIST\_POINTER\_ZERO



PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_STATUS\_ITEM\_LIST\_ZERO/SX\_STRATEGY\_DROP\_OLDEST  
 PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_STATUS\_ITEM\_NOT\_ZERO/SX\_STRATEGY\_DROP\_WRITE  
 PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_NOT\_AVAILABLE/SubscribeToDeviceStatusData  
 PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_NOT\_ZERO/TestSystem  
 PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_POINTER\_ZERO/SystemEx  
 PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VSX\_SYSTEM\_HANDLE\_ZERO/vsx\_UnsubscribeToDeviceStatusData  
 PF.VsxProtocolDriver.WrapperNE.h, [54](#)  
 VSX\_STATUS\_ERROR\_VSX\_TAG\_LIST\_NOT\_ZERO/vsx\_UploadData  
 PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_TAG\_LIST\_POINTER\_ZERO/vsx\_UploadParameterList  
 PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_TAG\_LIST\_ZERO/vsx\_UploadParameterSet  
 PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ERROR\_VSX\_VALUE\_POINTER\_ZERO/VsxCaptureInformation  
 PF.VsxProtocolDriver.WrapperNE.h, [57](#)  
 VSX\_STATUS\_ERROR\_XML\_COMMAND\_ZERO/VsxDataContainerHandle  
 PF.VsxProtocolDriver.WrapperNE.h, [55](#)  
 VSX\_STATUS\_ERROR\_XPATH\_ZERO/VsxDevice  
 PF.VsxProtocolDriver.WrapperNE.h, [56](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_BASE64/VsxDeviceList  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_BOOL/VsxDeviceStatusScope  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_DOUBLE/VsxDisconnectEvent  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_ENUM/VsxDisparityDescriptor2  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_FLOAT/VsxImage  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_HEXSTRING/VsxImageData2Format  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_INT/VsxLineCoordinate  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_INT16/VsxLineData  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_IP/VsxLineDataFormat  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_LONG/VsxOlr2CaptureInformation  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_POINT/VsxOlr2ModbusData  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_QUAD/VsxParameter  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_RECTANGLE/VsxParameterEnumItem  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_STRING/VsxParameterList  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_UINT/VsxParameterValueType  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_ITEM\_VALUE\_TYPE\_UNKNOWN/VsxSerialConnectionType  
 PF.VsxProtocolDriver.WrapperNE.h, [60](#)  
 VSX\_STATUS\_SUCCESS/VsxSessionTypes

- PF.VsxProtocolDriver.WrapperNE.h, [50](#)
- VsxStatusCode
  - PF.VsxProtocolDriver.WrapperNE.h, [49](#)
- VsxStatusItem
  - PF.VsxProtocolDriver.WrapperNE.h, [53](#)
- VsxStatusItemList
  - PF.VsxProtocolDriver.WrapperNE.h, [53](#)
- VsxStatusItemValueType
  - PF.VsxProtocolDriver.WrapperNE.h, [53](#)
- VsxStrategy
  - PF.VsxProtocolDriver.WrapperNE.h, [49](#)
- VsxSystemHandle
  - PF.VsxProtocolDriver.WrapperNE.h, [50](#)
- VsxTagList
  - PF.VsxProtocolDriver.WrapperNE.h, [52](#)
- VsxTransformation
  - PF.VsxProtocolDriver.WrapperNE.h, [51](#)
- width
  - \_VsxImage, [20](#)
  - \_VsxLineData, [23](#)
- x
  - \_VsxLineCoordinate, [21](#)
- y
  - \_VsxLineCoordinate, [21](#)
- z
  - \_VsxLineCoordinate, [21](#)