SENSOR TECHNOLOGY LTD

ORT/RWT Series Transducer DLL Programmer's Guide

STCOMMDLL_V5U

Revision 7 - December 2018

Table of Contents

Introduction **4**
Compatible Models **4**
Overview **5**
Initialising and Accessing a Transducer **5**
Discovering Transducers **6**
Data Block Functions **10**
Torque Types **10**
Peak Torque 10
Peak Torque with AutoReset 10
Peak Torque CW 11
Peak Torque CCW 11
PeakMinMax 11
Speed Modes **12**
Temperature Sensors **13**
Time Stamp **13**
Data Capture Mode **14**
Normal/Optimised 14
Emulated 15
DLL Dependencies **22**
DLL Type Definitions **22**
DLL Structures **23**
ST_DATABLOCK 23
MINMAX_TMP 25
CAPREC 25
VERSIONS 26
Status Codes (ST_STATUS) **27**
DLL Functions **28**
ST_DLL_Version 30
ST_How_Many_Ports 31
ST_Port_Name 31
ST_Find_Devices 32
ST_Find_Device_Status 33
ST_Find_Device_Result 34
ST_Find_Device_Terminate 34
ST_Open_Device 35
ST_Close_Device 35
ST_Close_ALL_Devices 35
ST_GETINFO_Model 36
ST_GETINFO_SerialNumber 36
ST_GETINFO_ID_String 37
ST_GETINFO_Manufacture_Date 38
ST_GETINFO_Calibration_Date 39
ST_GETINFO_Customer 40
ST_GETINFO_ConnectionMethod 41
ST_GETINFO_DeviceClass 42
ST_GETINFO_Firmware 43
ST_GETINFO_FirmwareEx 43
ST_GETINFO_FirmwareText 44
ST_GETINFO_FSD 45
ST_GETINFO_Units 46
ST_GETINFO_Max_Speed 47

ST_GETINFO_Speed_Gratings 48
ST_GET_Data_Block 49
ST_GET_Data_Block_Extract 50
ST_GET_Torque_Select 52
ST_GET_Torque_Select_Convert 53
ST_GET_Torque 54
ST_GET_Torque_Peak 55
ST_GET_Torque_Auto_Reset 55
ST_GET_Torque_Peak_MinMax 56
ST_GET_Speed_Fast 57
ST_GET_Speed_Slow 57
ST_GET_Power_In_Watts 58
ST_GET_Power_In_HP 59
ST_GET_Temperature_Ambient 60
ST_GET_Temperature_Shaft 60
ST_GET_Temperature_Internal 61
ST_GET_Torque_Filter 61
ST_SET_Torque_Filter 62
ST_GET_Speed_Filter 62
ST_SET_Speed _Filter 63
ST_RESET_Peaks 64
ST_Zero_Transducer 65
ST_ZeroAverage_Transducer 65
ST_Reset_TimeStamp 66
ST_GET_TimeStamp 66
ST_Capture_Enable 67
ST_Capture_Disable 68
ST_GET_Capture_Data 69
ST_GET_Capture_Rate 70

Contact Details

Sensor Technology Ltd,
Apollo Park,
Ironstone Lane,
Wroxton,
BANBURY,
OX15 6AY,
United Kingdom.

Sales
Email: stlsales@sensors.co.uk
Tel: +44 (0)1869 238400

Technical Support
Email: software@sensors.co.uk
Tel: +44 (0)1869 238400

Introduction

The DLL (dynamic link library) provides the programmer with a method of interfacing a program with an advanced ORT/RWT series transducer, without having to talk directly using the communication protocol.

The DLL simplifies the use of the USB and RS232 interfaces by providing a unified interface to access transducers connected via either method; it takes care of the low-level driver access, protocol negotiation and data manipulation.

Compatible Models

The DLL is compatible with transducers from the advanced ORT and RWT family of products. Transducers must be running firmware version 3 or higher and have digital communications enabled.

The table below lists the models that are compatible:

Transducer Family	**Model Range**	**Models**
Optical (ORT)	ORT240	ORT240/ORT241
Rayleigh Wave (RWT)	RWT320	RWT320/RWT321/RWT322
	RWT340	RWT340/RWT341/RWT342
	RWT420	RWT420/RWT421/RWT422
	RWT440	RWT440/RWT441/RWT442

Compatible transducers can be identified by the presence of a status LED and serial number greater than 12200.

Overview

The DLL was written to simplify and speed up the process of developing a custom application to interact with a transducer.

The DLL is written in C and can be used with a number of other programming languages. The function descriptions refer to C type variables, but equivalents can be used in other languages.

The functions available give access to most of the available data and control functions, the commands for accessing transducers connected by either RS232 or USB are the same.

The DLL can control up to 10 devices simultaneously, all the programmer need do is identify the required transducer by passing a device id with each function command.

Initialising and Accessing a Transducer

Before a transducer can be accessed the DLL needs to be initialised by finding connected transducers. When the DLL initialises it builds a device list in memory of the transducers that it finds. When a transducer is found the DLL downloads its configuration and saves the data with the transducer in the list. Device handles and connection settings are also saved. The list enables quick access to transducer configuration data and hides some of the complexity of accessing the underlying interfaces.

Transducers are opened, closed and accessed by providing a device id, the id is a reference to the device list, and uniquely identifies each connected transducer. Device id's start at 0, and are allocated incrementally as transducers are found.

The procedure for interacting with the DLL and subsequently the transducer is detailed below:

1. Initialise the DLL – Initialise the DLL by calling the **ST_Find_Devices** function, this will search the system for connected transducers and build up a list of the transducers found. The DLL will cache connection and configuration information from the transducers for local lookup.
2. Identify the attached transducer(s) (Optional) – Use the **ST_GETINFO** functions to identify and extract configuration information on the connected transducers.
3. Open a transducer – Use the **ST_Open_Device** function to select a transducer for use. The communication channel that the transducer is connected to is initialised and resources are allocated. Up to 10 transducers can be open for use at any time. A transducer will remain open until it is closed or the process calling the DLL exits.
4. Get data – Use the **ST_GET** and **ST_SET** functions to get data and configure settings on the selected transducer. Open transducers can be accessed at random.
5. Close a transducer – When access to a transducer is no longer needed, use the **ST_Close_Device** or **ST_Close_ALL_Devices** function to close the communication channel and free resources for the selected transducer. The DLL will automatically close all transducers when the calling process exits.

Discovering Transducers

The transducer discovery process is part of the initialisation phase and is initiated with the **ST_Find_Devices** function. There are two modes that can be used to control the discovery process, Mode 1 - Execute and wait and Mode 2 – Execute and return.

The amount of time required for the discovery process varies depending upon the number ports on the system and the search filter used. The search filter directs the discovery process on where it should look for transducers. The filter selected will have a big impact on how long the discovery process will take and should factor in the decision on what mode to use.

The first three filter options are always the same, no matter what configuration the system has. The table below lists the default options:

Filter Value	Filter	Description
0	All Devices	Search on all discovered interfaces.
1	RS232	Search on all available COM ports.
2	USB	Search on USB only.

If the system has COM ports all subsequent filter values map the COM ports in numerical order. The table below shows an example of how a system with three COM ports (COM2, COM4, and COM5) will map those ports to filter values:

Filter Value	Filter	Description
3	COM2	Search on COM2 only.
4	COM4	Search on COM4 only.
5	COM5	Search on COM5 only.

The DLL has a set of functions for notifying the calling program on the search options available. This functionality allows programmers to write an application that isn't PC specific. There are two parts to the process to get the list of filters, first get the list length using the **ST_How_Many_Ports** function, then retrieve each filter from the list using the **ST_Port_Name** function. When calling **ST_Port_Name** an index number is required to select between each filter, the index is referenced from 0 and is valid upto the list length. The index value used to retrieve the filters is also the filter value used to filter a search.

Transducers connected via USB are easily found as the operating system can be queried to see if there is a device attached. USB searches occur almost instantly. Transducers connected via RS232 are more difficult to detect, the detection process involves sending out a zero byte to request a transducer identification string, if the correct response is received the DLL will cache the transducers configuration and continue to the next search option as defined by the filter, if a timeout or a incorrect response occurs, the DLL will switch to a different baud rate and query the transducer again, this process repeats for each of the three baud rates available.

When RS232 is used as a search filter, it is recommended that mode 2 be used, as mode 1 can often cause your program to hang or receive the "not responding" message. If you specify a single COM port or only have one COM port and the transducer is connected to that port, then the mode does not matter as the transducer will be found almost instantly. The program hang will only be apparent when you have a COM port without a transducer attached.

Mode 1 – Execute and wait
The execute and wait mode initiates a DLL find device process and will block the calling process until a search has been completed. When the function call returns the process will have completed and is ready for the next function call. The execute and wait mode is selected by setting the waitforcomplete parameter to TRUE.

This mode requires only a single call, if the connection method is known and the filter is selected accordingly, then mode 1 is the best option.

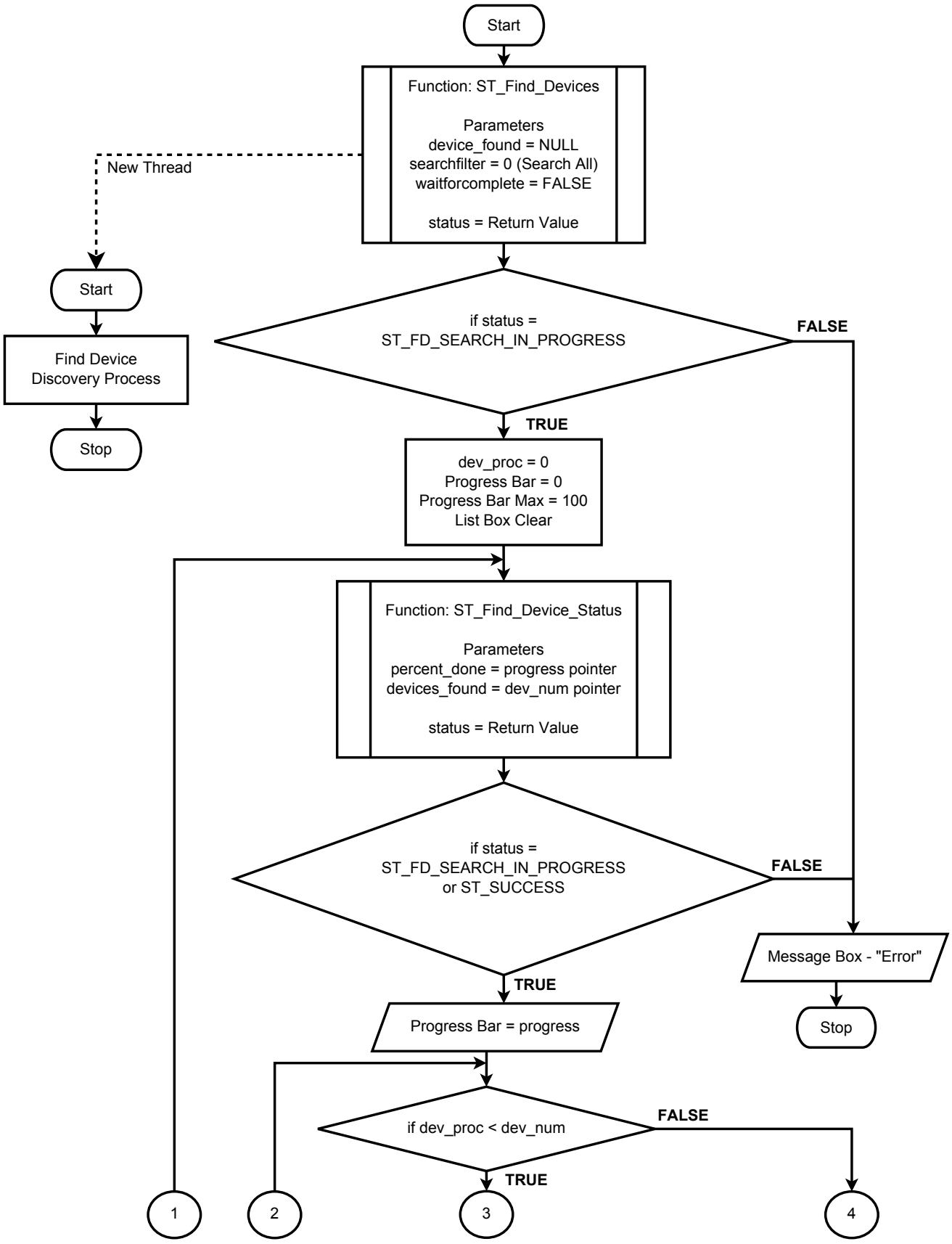
Mode 2 – Execute and return
The execute and return mode initiates a DLL find device process and returns execution back to the calling process. When the find device process initiates it spawns a new thread where the search is run. Until the process completes you cannot open or access a transducer.

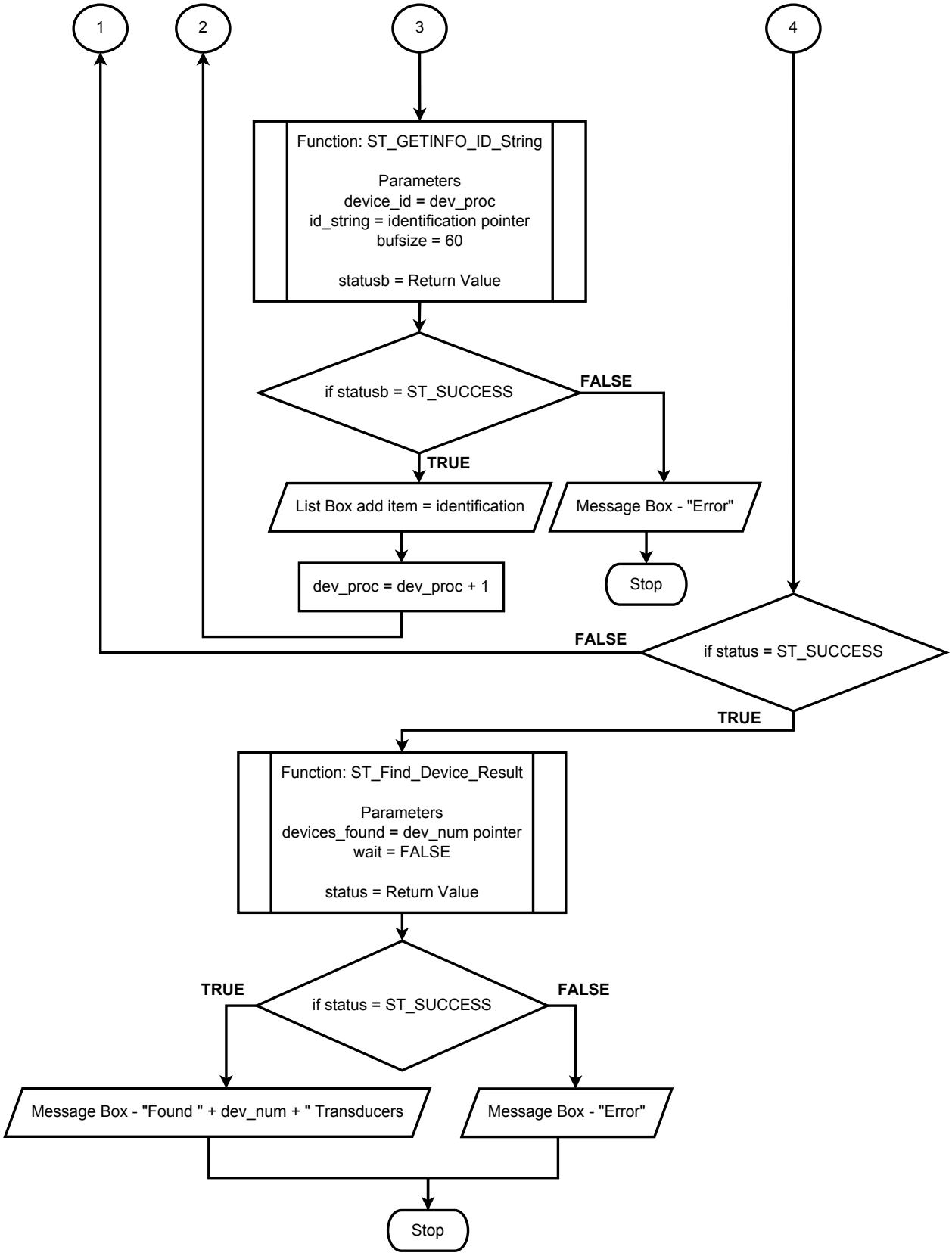
While the find device process is running it will need to be monitored for progress and completion. Process monitoring is accomplished by using the **ST_Find_Device_Status** function, which will provide information on the progress percentage and the number of devices found. The **ST_Find_Device_Status** function will return success when the find device process has completed, when this occurs call the **ST_Find_Device_Result** function to deallocate used resources and enable devices to be openned.

While the find device process is being monitored using **ST_Find_Device_Status**, the **GET_INFO** commands can be used to retrieve information on transducers that have been found.

The execute and return mode is selected by setting the waitforcomplete parameter to FALSE.

The flow chart below shows an example of how to use the discovery process in mode 2. The example is based on a simple form with a progress bar and list box.




1
2
3
4
Function: ST_GETINFO_ID_String
Parameters
device_id = dev_proc
id_string = identification pointer
bufsize = 60
statusb = Return Value
if statusb = ST_SUCCESS
FALSE
TRUE
List Box add item = identification
Message Box - "Error"
dev_proc = dev_proc + 1
Stop
FALSE
if status = ST_SUCCESS
TRUE
Function: ST_Find_Device_Result
Parameters
devices_found = dev_num pointer
wait = FALSE
status = Return Value
TRUE
if status = ST_SUCCESS
FALSE
Message Box - "Found " + dev_num + " Transducers
Message Box - "Error"
Stop

Data Block Functions

The DLL includes two block capture functions which transfer complete data sets from the connected transducer. These functions enable convenient access to all available data with a single function call.

The down side of using these block commands is the amount of data that is transferred during the transaction. The transfer time required will significantly affect the maximum achievable capture rate. If sample rate is important, you may wish to consider using the data capture mode, alternatively the single data request functions can be used.

Torque Types

The primary purpose of the ORT/RWT series transducers is to measure torque, the torque value that is output from the transducer is run through several processes within the firmware, these processes include a filter (if enabled), frequency to torque rescaling, temperature correction and zero offset adjustment. The filter is a running average with a standard deviation cut off to remove spurious readings, the running average enables the sample throughput to be unaffected by filter size.

Once the final torque value is computed it is run though a peak torque capture algorithm. The peak torque algorithm monitors the incoming data and compares it against a set of stored values using various criteria. If the value matches the criteria, that value replaces the stored value. In most cases the criterion is related to whether the captured value is greater than the stored value.

Peak values assume a reset position on start-up, when peak values are reset they are set to zero, PeakMinMax values are set to the current torque value.

The peak torque algorithm is run on every valid torque reading captured, ensuring that no peak value is missed.

The torque value unless specified will always be scaled in the native unit of measurement for the transducer.

The following subsections describe the different types of peak torque.

Peak Torque

The peak torque value indicates the highest torque applied to the transducer in either direction. The value is signed to indicate the direction that the torque was applied in.

Peak Torque with AutoReset

The peak torque with auto reset is similar to the peak torque feature, it works in the same way by recording the maximum torque, but automatically resets to zero when the current torque value drops below a configured percentage of the peak value. When the reset triggers the peak is held for a few seconds before it is zeroed.

The default auto reset percentage is 80%; the percentage can be configured using the "Transducer Control Utility", which accompanies our advanced transducers.

Peak Torque CW
The peak torque CW value records the highest torque value measured in the clockwise direction.

Peak Torque CCW
The peak torque CCW value records the highest torque value measured in the counter-clockwise direction.

PeakMinMax
The PeakMinMax feature monitors the captured torque values and records the lowest and highest value from a reference position. This reference is given via the **ST_RESET_Peaks** command and assumes zero on power on. An example of the PeakMinMax feature is as follows: if the reference is set to 10, then the torque value goes up by 10 and down by 12, Max would be 20 and Min would be -2.

Speed Modes

Speed is decoded from a square wave signal, produced by a shaft mounted grating passing through an optical sensor. The frequency of the square wave indicates the rotational speed of the shaft. The transducer uses two methods for the measurement of speed, both methods run simultaneously, offer good accuracy, but differ in measurement time. Speed is always measured in revolutions per minute (RPM).

Slow – The slow method uses a frequency count. Rising edges of the square wave are counted over a period of a second, and then calculated into RPM. As the name suggests this method is slow as measurements will be produced at a rate of 1 a second. This method is good if you have a fluctuating drive speed and wish to filter the captured speed value.

Fast – The fast method uses a period count. The period count measures the time between rising edges of the square wave, then computes the RPM by turning the time into frequency. The fast methods measurement rate is variable and is directly related to the rotational speed of the transducer. When the rotational speed of the shaft rises above 2000 RPM the fast method will increase the number of rising edges over which time is measured, this is done to preserve measurement accuracy.

The fast methods measurement rate can be calculated from the following tables. The measurement rate differs between the 300 series and 400 series because of a different implementation. The calculations shown are based on a standard 60 line grating.

RWT320/340 (MKII)

Rotational Speed (RPM)		**Update Rate (Hz)**
From	**To**	
0		1 Hz
1	2000	RPM / 2
2000	4000	((RPM – 2000) x 0.3227) + 650
4000	8000	((RPM – 4000) x 0.196) + 800
8000	16000	((RPM – 8000) x 0.1117) + 850
16000	32000	((RPM – 16000) x 0.058) + 900

ORT240/RWT420/440 (MKIII)

Rotational Speed (RPM)	**Update Rate (Hz)**
0	1 Hz
< 2000	RPM
> 2000	RPM x (1 / (⌊ (RPM - 1) / 2000 ⌋ + 1))

Both modes have there own peak monitor to record the highest measured speed.

Temperature Sensors

The transducer monitors temperature from three different sensors, these are defined as ambient, shaft and internal. The shaft temperature is the only one which is used for compensation; the other two are for monitoring only. The transducer measures temperature in degrees Celsius.

> ***Ambient*** – The ambient sensor is mounted in free air, stood off from the PCB it is mounted to.
>
> ***Shaft*** – The shaft sensor is an infra-red device that is pointed directly at the centre of the shaft.
>
> ***Internal*** – The internal sensor is part of the communications processor on the main processing board.

Some board revisions may not feature all three sensors. In a case where the sensor is missing, the sensor value will be taken from another sensor. The shaft sensor is always present.

Time Stamp

The DLL has a primitive time stamping system based on elapsed time in milliseconds; the elapsed time is counted from a fixed point controlled by the user. The time stamp is not directly linked to the transducer readings, but should give an approximate correlation between the reading and time, provided that the commands to request data and the time stamp are called together.

The start point from which the elapsed time is counted is controlled by the **ST_Reset_TimeStamp** function. The elapsed time from the start point can be retrieved using the **ST_GET_TimeStamp** function. If the time stamp is not initialised, any call to get the elapsed time will return zero.

Data Capture Mode

The data capture mode is a mechanism which automatically captures data from a connected transducer at a user specified rate. Data can be captured at rates anywhere from 1 capture per second to up to 50000 captures per second.

The main purpose of the data capture mode is to extract data from a transducer at high capture rates. High capture rates are only possible when using USB and through a combination of specially optimised hardware and firmware. These optimisations are only present on ORT/RWT transducers with firmware greater than 5. For non-optimised transducers or when using RS232, the data capture mode will operate in an emulation mode.

The operating system timers must be capable of a 1ms resolution for the capture system to work. The DLL will request the timer resolution on initialisation and request a 1ms resolution.

The following table outlines the differences in capture rate between normal/optimised and emulation mode.

<table>
<tr><th>Mode</th><th>Connection Method</th><th>Baud Rate</th><th>Maximum Capture Rate (Records Per Second)</th></tr>
<tr><td>Normal/Optimised ORT Series</td><td rowspan="3">USB</td><td rowspan="3">12 Mbps</td><td>Up to 50000</td></tr>
<tr><td>Normal/Optimised RWT Series</td><td>Up to 10000</td></tr>
<tr><td rowspan="4">Emulated</td><td>200</td></tr>
<tr><td rowspan="3">RS232</td><td>9600 bps</td><td>5</td></tr>
<tr><td>38400 bps</td><td>10</td></tr>
<tr><td>115200 bps</td><td>50</td></tr>
</table>

The following sections give a brief overview of how the modes work.

Normal/Optimised – (USB and Firmware Version 5 or greater)

The optimised mode is used when the data capture is activated with a compliant transducer using USB.

The optimised mode achieves its higher capture rates by fully utilising the bandwidth available on the USB bus. In order to achieve this, a bulk packeted buffered approach is required.

When the data capture mode is activated the DLL tells the transducer to initiate a continuous self timed capture at the users requested rate. The captured data is written to one of sixteen buffers. When a buffer is full it is marked for transfer, If no transfer is active a transfer is initiated. If a buffer is

filled before the previous buffer transfer has been initiated, the newly filled buffer is chained to the previous buffer. The sixteen buffer design should protect against data loss should the DLL be starved of CPU time or the USB bus is busy. The transducer will automatically deactivate the capture mode if all sixteen buffers become filled and a seconds worth of data is lost.

Each USB transfer is 836 bytes in size and uses a dedicated transmission pipe, the use of a secondary pipe allows the transducer to respond to data requests in the normal manner. At high data rates the transducer should not be interupted with any other data request.

Each data transfer contains a header and 100 record sets. The header contains a base time stamp, record time increments, temperature readings and slow speed reading. A record set contains a single torque and fast speed reading.

The maximum capture rate achievable will depend upon the transducer technology and transducer tuning. RWT transducers with an optimal setup can achieve a capture rate of 10KHz. Shipped transducers are typically around 5KHz to 6KHz. ORT transducers are capable of capturing data at 50KHz.

To establish the capture rate capability of a transducer, use the **ST_GET_Capture_Rate** function, which returns the capture rate measured by the transducer.

When using this mode be aware that USB is a host controlled bus based architecture and can be effected by other bus traffic or host activity. The capture rate requested should consider the amount of data that will be generated, the number of active USB devices and the load on the host PC. It may be necessary to reduce the capture rate to achieve reliable operation.

In this mode, the capture rate and timestamp are generated from the transducers core clock. Any variance in the core clock will cause the data capture and time stamp to slide from real time, the per reading error will be very small, but over millions of cycles it may slide out.

Emulated – (RS232 or non-optimised transducers)
The emulated mode is used when the connected transducer is using RS232 or does not have the right firmware.

Emulated mode polls the transducer for data using the **ST_GET_Data_Block** function. The timestamp is applied when the data is received and is the elapsed time in microseconds from activation.

Data capture is triggered by a waitable timer, whose accuracy, resolution and prompt execution isn't guaranteed. In this mode, data capture is at the mercy of the Windows scheduler, to improve execution success the DLL raises the capture thread priority level.

Data Capture Mode Operation

The data capture mode operates within its own thread and its priority is elevated to ensure CPU access. Data captures are triggered by a timer which is configured to trigger at the capture frequency. Captured data is inserted into a ring buffer which can accessed by both the capture thread and the users thread.

The ring buffer is made up of 262,144 ***CAPREC*** records (refer to the DLL Structures section for a definition). Each record contains information on one category of data, e.g. Torque, Speed, or Temperature.

The table below shows the record layout.

Field	**Description**
Time	Record timestamp which is the offset in milliseconds from start.
Type	Type identifies the category of data.
Value	Data value.

The type field is a numeric value, the table below identifies the different categories and there corresponding key values.

Type	**Key Value**	**DLL Definition**
Torque	0	CAPREC_TORQUE
Speed (Fast)	1	CAPREC_SPEED_FAST
Speed (Slow)	2	CAPREC_SPEED_SLOW
Temperature (Shaft)	3	CAPREC_TEMP_SHAFT
Temperature (Ambient)	4	CAPREC_TEMP_AMB

The ring buffer is accessed using a read and write pointer, it is important that the user reads the buffer frequently enough so as to avoid the head reaching the tail. If the buffer becomes full the data capture will terminate.

The data capture mode is activated by calling the **ST_Capture_Enable** function. The enable function requires the user to specify the capture rate that they require. The value can be anything between 1 and the maximum capture rate available. The **ST_GET_Capture_Rate** function should be used to retrieve the maximum.

The capture rate specifies the rate at which the transducer will be polled or requested to capture data. The capture rate does not directly indicate how many records will be generated per second.

The following calculations can be used to calculate the number records that will be generated from a given capture rate.

Normal/Optimised

$$Records = \left(\left\lceil \frac{Capture\ Rate}{100} \right\rceil \times 3 \right) + \left(Capture\ Rate \times 2 \right)$$

Emulated

$$Records = Capture\ Rate \times 5$$

Data is transferred from the ring buffer by using the **ST_GET_Capture_Data** function. The user is required to pass an array of ***CAPREC*** records and notify the function of the array depth. The function copies records from the ring buffer to the array until the array is full or there are no further records to be copied. On function completion the number records written to the array will be returned.

The array depth, **ST_GET_Capture_Data** call frequency and capture rate should all be considered when writing a program which uses the data capture mode. There is some overhead in ensuring thread syncronisation so a small array and high call frequency is not recommended.

The following table shows a sample set of data captured from a transducer. The capture is using the optimised mode and configured to run at 5Hz. The table shows 2 seconds of data.

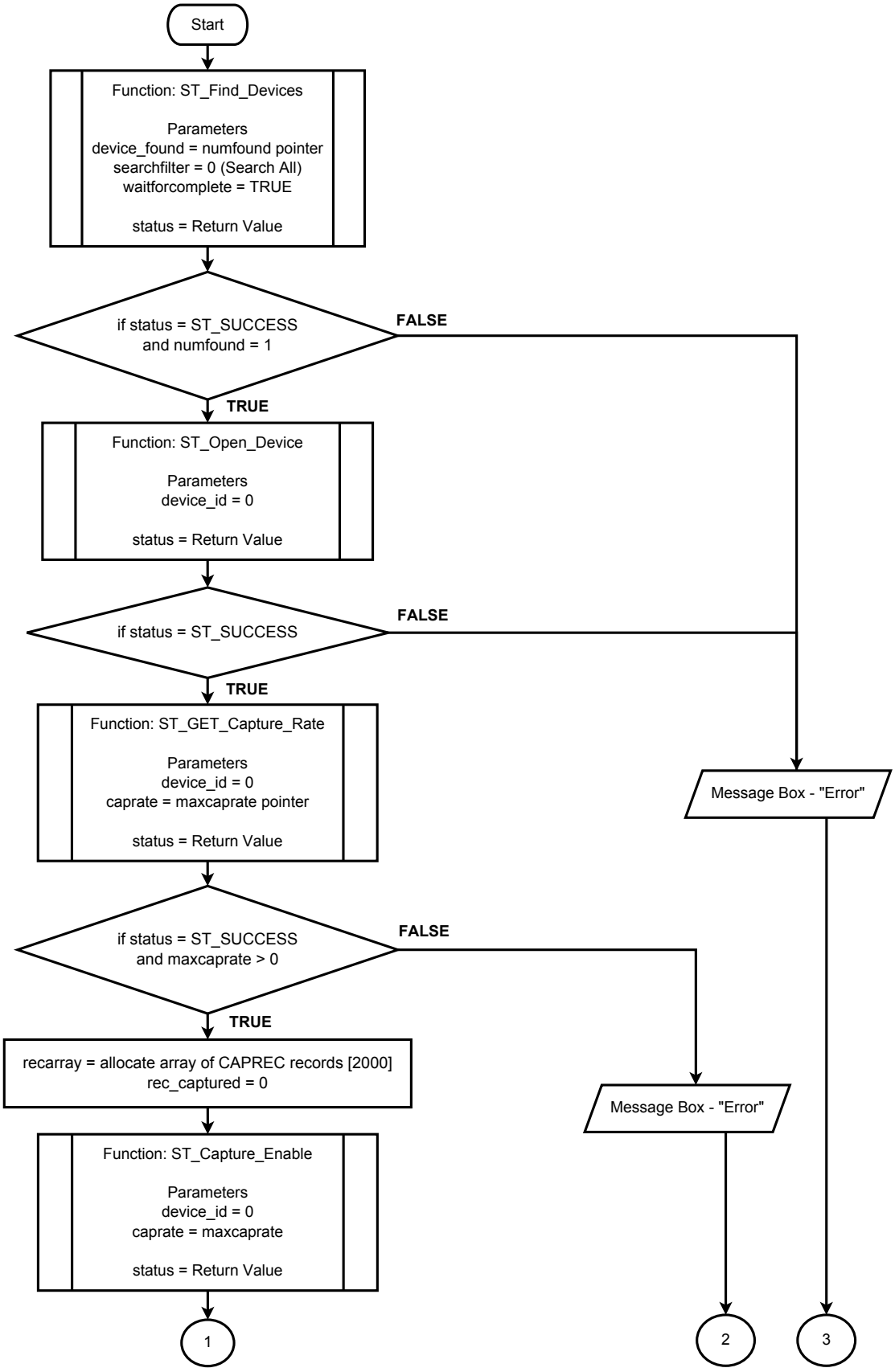
Time	Type	Value
0	2	0
0	3	25
0	4	25
0	0	0.1
0	1	0
200	0	0.1
200	1	0
400	0	0.1
400	1	0
600	0	0.1
600	1	0
800	0	0.1
800	1	0
1000	2	0
1000	3	25
1000	4	25
1000	0	0.1
1000	1	0
1200	0	0.1
1200	1	0
1400	0	0.1
1400	1	0
1600	0	0.1
1600	1	0
1800	0	0.1
1800	1	0

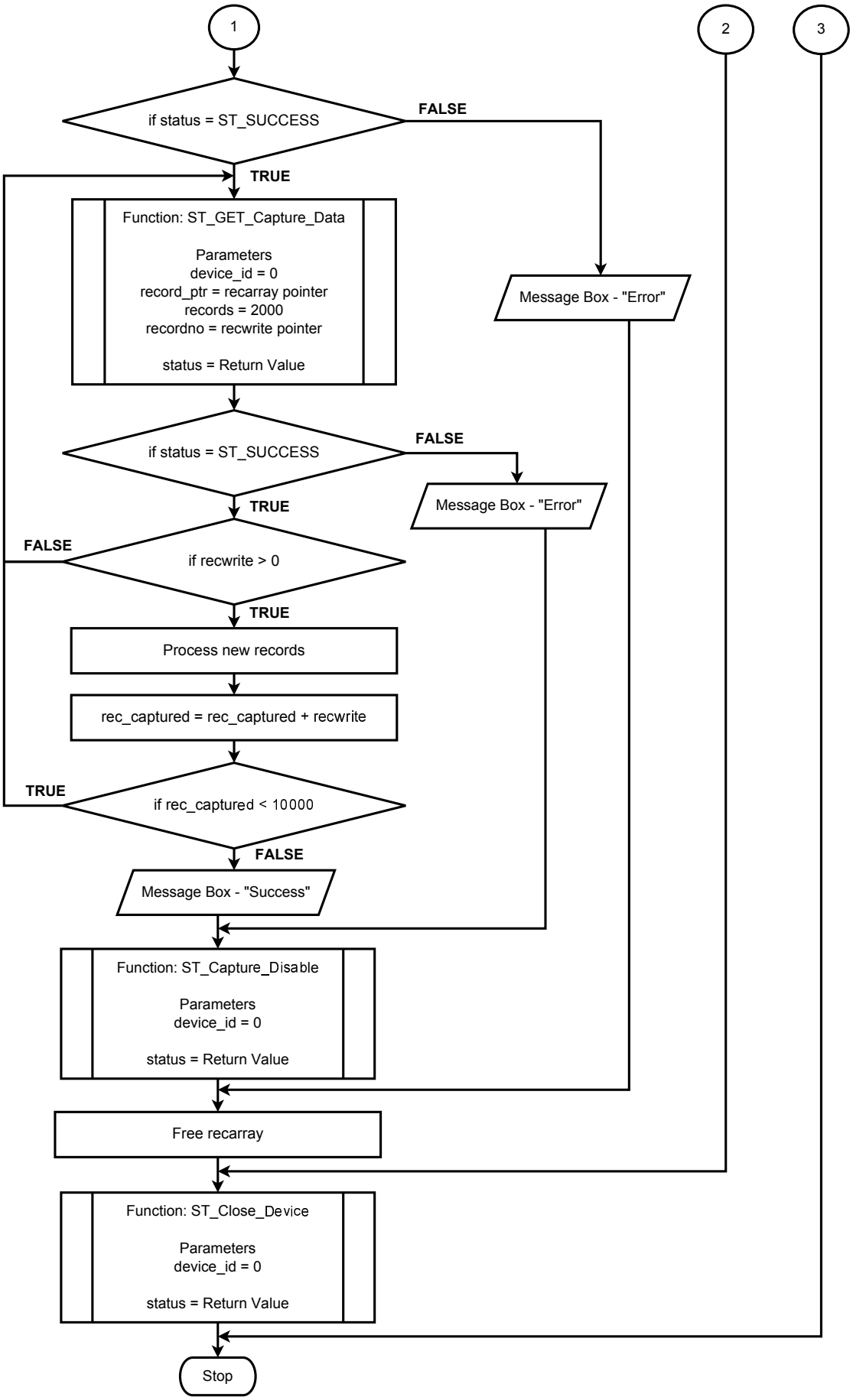
The following table shows a sample set of data captured from a transducer. The capture is using the emulated mode and configured to run at 5Hz. The table shows 1 second of data.

Time	Type	Value
0	0	0.1
0	1	0
0	2	0
0	3	25
0	4	25
200	0	0.1
200	1	0
200	2	0
200	3	25
200	4	25
400	0	0.1
400	1	0
400	2	0
400	3	25
400	4	25
600	0	0.1
600	1	0
600	2	0
600	3	25
600	4	25
800	0	0.1
800	1	0
800	2	0
800	3	25
800	4	25

When the capture mode is no longer required it should be stopped so that its resources can be released. To stop an active capture call the **ST_Capture_Disable** function. Once called the capture thread will be stopped and the ring buffer will be cleared and released.

The flow chart below demonstrates a simple program which uses the data capture mode to capture 10000 records at the maximum capture rate.




1
2
3
if status = ST_SUCCESS
FALSE
TRUE
Function: ST_GET_Capture_Data
Parameters
device_id = 0
record_ptr = recarray pointer
records = 2000
recordno = recwrite pointer
status = Return Value
Message Box - "Error"
if status = ST_SUCCESS
FALSE
TRUE
Message Box - "Error"
FALSE
if recwrite > 0
TRUE
Process new records
rec_captured = rec_captured + recwrite
TRUE
if rec_captured < 10000
FALSE
Message Box - "Success"
Function: ST_Capture_Disable
Parameters
device_id = 0
status = Return Value
Free recarray
Function: ST_Close_Device
Parameters
device_id = 0
status = Return Value
Stop

DLL Dependencies

The DLL was built using Microsoft Visual Studio 2013 and requires the 2013 C Runtime Libraries. The DLL is dynamically linked to the runtime and will need to have the appropriate 2013 C Runtime Libraries installed. The CRT redistributable can be downloaded from Microsoft (https://www.microsoft.com/en-gb/download/details.aspx?id=40784).

In addition to the runtime library, a USB driver library (libusb0.dll) must be present. The library allows the DLL to interface with the transducers USB driver. Ensure that the USB DLL is accessible by STCOMMDLL_V5U.dll, either by being in the same directory or accessible via the PATH environmental variable. Older versions of the DLL required different USB drivers and dependencies. DLL version 4.1.8 switches to a unified USB driver, which supports both MKII and MKIII transducers.

DLL Type Definitions

A number of custom defined variable types have been used in the DLL functions, the non-standard types have been defined in the table below:

Type Definition	Data Type
ST_STATUS	unsigned long (4 bytes)
UCHAR	unsigned char (1 byte)
UINT16	unsigned short / int (2 bytes)
DWORD	unsigned long (4 bytes)
INT32	long (4 bytes)
BOOL	unsigned long (4 bytes)
UINT64	unsigned long long (8 bytes)

Most of the above types are defined in the Windows API.

BOOL – The table below shows that mapping between boolean value and numeric value.

Boolean Value	Numeric Value
TRUE	1
FALSE	0

DLL Structures

Structures are used frequently within the DLL to pass multiple related variables in a single block and passed in a single parameter.

ST_DATABLOCK

The ST_DATABLOCK type is a structure that contains a complete transducer data set. The data set contains all the data that can be captured from the transducer.

The structure is split up into sub structures, each one related to a specific category of data.

Torque

The torque values use the native unit of measurement of the Transducer.

Type	Name	Description
float	Torque	Current torque value.
float	Torque_Peak	Peak torque value.
float	Torque_Auto_Reset	Peak torque value with auto reset.
float	Torque_Peak_CW	Peak Torque value in the CW direction.
float	Torque_Peak_CCW	Peak Torque value in the CCW direction.
MINMAX_TMP	MinMax	Lowest/Highest torque value.

Speed (slow/fast modes explained in the speed section).

All measurements are output in RPM.

Type	Name	Description
INT32	Slow	Current speed value from the slow capture mode.
INT32	Fast	Current speed value from the fast capture mode.
INT32	Slow_Peak	Peak speed value from the slow capture mode.
INT32	Fast_Peak	Peak speed value from the fast capture mode.

Angle

Type	Name	Description
INT32	Rotations	Shaft rotations.
INT32	Degrees	Shaft rotation in degrees from reset point/last rotation.

If an Angle sensor is not fitted, the speed pulses are used to generate the rotations and degrees in a incremental mode. The accuracy in this mode may be affected by external conditions.

Power (slow/fast modes explained in the speed section).

Type	Name	Description
float	Watts_Slow	Power in watts calculated from the last torque and slow speed value.
float	Watts_Fast	Power in watts calculated from the last torque and fast speed value.
float	Peak_Watts_Slow	Peak power in watts, based on the speed from the slow capture mode.
float	Peak_Watts_Fast	Peak power in watts, based on the speed from the fast capture mode.
float	HP_Slow	Power in HP calculated from the last torque and slow speed value.
float	HP_Fast	Power in HP calculated from the last torque and fast speed value.
float	Peak_HP_Slow	Peak power in HP, based on the speed from the slow capture mode.
float	Peak_HP_Fast	Peak power in HP, based on the speed from the fast capture mode.

Temperature (degrees Celsius)

Type	Name	Description
float	Ambient	Ambient temperature.
float	Shaft	Shaft temperature.

MINMAX_TMP

The MINMAX_TMP structure is comprised of min and max torque variables.

Type	Name	Description
float	max	Maximum torque value from reference/reset.
float	min	Minimum torque value from reference/reset.

CAPREC

The CAPREC structure is used with the embedded data capture mode.

Type	Name	Description
UINT64	time	Elapsed time in microseconds (µs).
UINT32	type	Data identifer. Type defines what value is, e.g. Torque/Speed. Refer to the Data Capture Mode section for a table of values.
float	value	Captured reading.

VERSIONS

The VERSIONS structure contains the transducer firmware revision. For firmware earlier than 5, the revision is converted from a simpler M.m format and the build is 1.

Type	**Name**	**Description**
UINT32	firm_type	Internal use descriptor
UINT16	firm_rev	Fimware revision in Binary-Coded Decimal. Format (Hex): 0xMMms M = Major m = Minor s = Sub Minor Example: 0x0122 = 1.2.2
UINT16	firm_build	Firmware build number

Status Codes (ST_STATUS)

Most functions defined in the DLL return a ST_STATUS message; this message acknowledges either a successful execution or a failure of some kind.

The table below lists the status codes and associated messages, the status codes are defined in the DLL header file as listed:

Status Code	Status Message	DLL Definition
0	Success	ST_SUCCESS
1	Busy	ST_BUSY
2	Command Active	ST_CMD_ACTIVE
3	Command Pending	ST_CMD_ACTIVE_PENDING
4	Command Complete	ST_CMD_ACTIVE_COMPLETE
5	Command Inactive	ST_CMD_INACTIVE
6	Failure	ST_FAILURE
7	Device Not Open	ST_DEVICE_NOT_OPEN
8	Checksum Error	ST_CHECKSUM_ERROR
9	Device Invalid	ST_DEVICE_INVALID
10	Buffer Too Small	ST_BUFFER_TOO_SMALL
11	Not Available In Firmware	ST_NOT_AVAILABLE_WITH_FIRMWARE
12	No Communications In Progress	ST_NO_COMMS_IN_PROCESS
13	Search In Progress	ST_FD_SEARCH_IN_PROGRESS
14	Too Many Requests	ST_TOO_MANY_REQUESTS
15	Access Violation	ST_ID_VALID_ACCESS_VIOLATION
16	Feature Not Fitted	ST_FEATURE_NOT_FITTED
17	Parameter Error	ST_PARAMETER_ERROR
19	Speed not fitted.	ST_SPEED_NOT_FITTED
20	Analog selection invalid.	ST_ANALOG_NOT_SELECTED
21	Analog channel not calibrated.	ST_ANALOG_NOT_CALIBRATED
22	Internal Flag	ST_FD_TERMINATED
23	Internal Buffer Overflow	ST_BUFFER_OVERFLOW
24	1ms timer resolution not possible	ST_WINDOWS_TIMER_RESOLUTION

DLL Functions

The DLL provides access to most of the available data and control features of the attached transducers. The following table summarises each of the DLL functions.

Functions	Function Description
ST_DLL_Version	DLL code version.
ST_How_Many_Ports	Get the filter list length.
ST_Port_Name	Get filter name for index.
ST_Find_Devices	Initialise DLL and initiate find device process.
ST_Find_Device_Status	Get the status of the running find device process.
ST_Find_Device_Result	Get the result of the find device process.
ST_Find_Device_Terminate	Terminate a find device process.
ST_Open_Device	Open a transducer for use.
ST_Close_Device	Close an open transducer.
ST_Close_ALL_Devices	Close all open transducers.
ST_GETINFO_Model	Get transducer model number.
ST_GETINFO_SerialNumber	Get transducer serial number.
ST_GETINFO_ID_String	Get transducer ID string (model, serial, firmware).
ST_GETINFO_Manufacture_Date	Get transducer manufacture date.
ST_GETINFO_Calibration_Date	Get transducer calibration date.
ST_GETINFO_Customer	Get the registered customer name.
ST_GETINFO_ConnectionMethod	Get the connection method for the attached transducer.
ST_GETINFO_DeviceClass	Get transducer technology class.
ST_GETINFO_Firmware	Get transducer firmware revision (legacy).
ST_GETINFO_FirmwareEx	Get detailed transducer firmware revision.
ST_GETINFO_FirmwareText	Decode transducer firmware revision.
ST_GETINFO_FSD	Get transducer FSD value.
ST_GETINFO_Units	Get transducer native unit of measurement.
ST_GETINFO_Max_Speed	Get transducer maximum rated speed.
ST_GETINFO_Speed_Gratings	Get transducer grating size.
ST_GET_Data_Block[1]	Get transducer data set.
ST_GET_Data_Block_Extract[1]	Get transducer data set.
ST_GET_Torque_Select[1]	Get selected torque value.
ST_GET_Torque_Select_Convert[1]	Get selected torque value and convert to unit.
ST_GET_Torque[1]	Get current torque value.
ST_GET_Torque_Peak[1]	Get peak torque value.

ST_GET_Torque_Auto_Reset[1]	Get peak torque with auto reset value.
ST_GET_Torque_Peak_MinMax[1]	Get PeakMinMax value.
ST_GET_Speed_Fast[1]	Get fast mode speed value.
ST_GET_Speed_Slow[1]	Get slow mode speed value.
ST_GET_Power_In_Watts[1]	Get current power in Watts.
ST_GET_Power_In_HP[1]	Get current power in HP.
ST_GET_Temperature_Ambient[1]	Get ambient temperature.
ST_GET_Temperature_Shaft[1]	Get shaft temperature.
ST_GET_Temperature_Internal[1]	Get internal sensor temperature.
ST_GET_Torque_Filter[1]	Get current torque filter setting.
ST_SET_Torque_Filter[1]	Set torque filter setting.
ST_GET_Speed_Filter[1]	Get current speed filter setting.
ST_SET_Speed_Filter[1]	Set speed filter setting.
ST_RESET_Peaks[1]	Reset torque, speed, power peaks.
ST_Zero_Transducer[1]	Zero transducer torque value.
ST_ZeroAverage_Transducer[1]	Zero transducer with an averaged torque value.
ST_Reset_TimeStamp	Initialise/reset timestamp to zero.
ST_GET_TimeStamp	Get elapsed time.
ST_Capture_Enable[1]	Enable capture mode.
ST_Capture_Disable[1]	Disable capture mode.
ST_GET_Capture_Data[1]	Get captured data.
ST_GET_Capture_Rate[1]	Get maximum capture rate.

[1] Transducer must be open to use command.

ST_DLL_Version

The **ST_DLL_Version** function returns the version and build of the DLL.

```
Void ST_DLL_Version(
        DWORD *version,
        DWORD *build
        );
```

Parameters

version	pointer to a variable of type DWORD that receives the DLL version.
build	pointer to a variable of type DWORD that receives the DLL build number.

Return value

None

Remarks

The format of the version number is in the format major.minor. The upper nibble of the lowest byte is the major version, while the lower nibble of the lowest byte is the minor version.

ST_How_Many_Ports
The **ST_How_Many_Ports** function returns the number of searchable options/filter items.

```
ST_STATUS ST_How_Many_Ports(
        DWORD *portcount
        );
```

Parameters

portcount	pointer to a variable of type DWORD that receives the number of items in the filter list.

Return value
If successful the function will return ST_SUCCESS, if an error occurs ST_FAILURE will be returned.

Remarks
Use the **ST_PORT_NAME** function to retrieve the name of each filter item, list indexes are valid upto the value of portcount.

ST_Port_Name
The **ST_Port_Name** function returns the filter name for the requested filter index.

```
ST_STATUS ST_Port_Name(
        DWORD portref,
        char *port_string,
        DWORD bufsize
        );
```

Parameters

portref	index of the filter name to retrieve, valid indexes are from 0 to the number of filters in the list.
port_string	pointer to an array of characters to receive the filter name. The maximum string length that will be returned is 20 characters long.
bufsize	size of the buffer passed in the port_string parameter.

Return value
If successful the function will return ST_SUCCESS, if an error occurs ST_FAILURE will be returned.

Remarks
The returned string will be null terminated.

ST_Find_Devices

The **ST_Find_Devices** function initialises the DLL and searches the system for connected transducers; it builds a list of connected transducers and caches configuration and connection information for each.

```
ST_STATUS ST_Find_Devices(
        DWORD *devices_found,
        DWORD searchfilter,
        BOOL waitforcomplete
        );
```

Parameters

devices_found	pointer to a variable of type DWORD that receives the number of transducers found.
searchfilter	selects the interfaces/ports that the find device process should use to find connected transducers.
waitforcomplete	boolean value to control the execution mode of the find device process. If TRUE, mode 1 - execute and wait will be selected. If FALSE, mode 2 – execute and return will be selected. Refer to the Discovering Transducers section for more information.

Return value

If successful the function will return ST_SUCCESS, if the function was called with the waitforcomplete parameter set to FALSE the function will return ST_FD_SEARCH_IN_PROGRESS. If an error occurs ST_FAILURE will be returned.

Remarks

When using mode 2 - execute and return (waitforcomplete = FALSE) use the **ST_Find_Device_Status** function to monitor the progress of the find device process, when complete use the **ST_Find_Device_Result** function to complete the discovery process. The **ST_Find_Device_Result** function is automatically called when using mode 1 - execute and wait (waitforcomplete = TRUE).

ST_Find_Device_Status

The **ST_Find_Device_Status** function returns the status of a find device process.

```
ST_STATUS ST_Find_Device_Status (
        DWORD *percent_done,
        DWORD *devices_found
        );
```

Parameters

percent_done	pointer to a variable of type DWORD that receives the find device progress in percent.
devices_found	pointer to a variable of type DWORD that receives the number of transducers found by the find device process.

Return value

If a transducer search has completed the function will return ST_SUCCESS, if a transducer search is still in progress ST_FD_SEARCH_IN_PROGRESS will be returned. If an error has occurred ST_FAILURE will be returned.

Remarks

The ideal way to use the **ST_Find_Device_Status** function is to display the progress percentage on a progress bar and as transducers are found use the **GET_INFO** functions to display identification information.

ST_Find_Device_Result
The **ST_Find_Device_Result** function completes the find device process initiated by **ST_Find_Devices**. The function will deallocate the resources used by the search and enables devices to be openned.

```
ST_STATUS ST_Find_Device_Result(
        DWORD *devices_found,
        BOOL wait
        );
```

Parameters

devices_found	pointer to a variable of type DWORD that receives the final number of transducers found by the search process. devices_found is only valid when the return value is ST_SUCCESS.
wait	boolean value to select whether the function should wait if a find device process is still in progress. If TRUE the function will block until the find device process has completed, if FALSE and a search is still in progress ST_FD_SEARCH_IN_PROGRESS will be returned.

Return value
If a find device process has completed successfully ST_SUCCESS will be returned, if a search process is still in progress ST_FD_SEARCH_IN_PROGRESS will be returned, if an error occurs ST_FAILURE will be returned.

Remarks
ST_Find_Device_Result only needs to be called when using the find device process in execute and return mode 2 (**ST_Find_Devices –** waitforcomplete = FALSE) and ideally when the **ST_GET_Process_Status** indicates completion. When using the find device process in execute and wait mode 1, the **ST_Find_Device_Result** function is automatically called.

ST_Find_Device_Terminate
The **ST_Find_Device_Terminate** function terminates an active find device process. The function will deallocate the resources used by the search and clear the transducer list of all entries.

```
ST_STATUS ST_Find_Device_Terminate (void);
```

Parameters
None

Return value
If a find device process was successfully terminated the function will return ST_SUCCESS, if no search was active the function will return ST_CMD_INACTIVE, if an error occurs ST_FAILURE will be returned.

Remarks
A find device termination request can take upto 4 seconds to complete, during this time the calling process will be blocked. The function will first attempt to stop the process cleanly by triggering an event, if the thread does not finish within 4 seconds it is terminated.

ST_Open_Device
The **ST_Open_Device** function opens a transducer for use with the DLL.

```
ST_STATUS ST_Open_Device(
        DWORD device_id
        );
```

Parameters
device_id device id of the transducer to open, id's are indexed from 0 up to the number of transducers found.

Return value
If a transducer is opened successfully the function will return ST_SUCCESS, if the device_id is invalid ST_DEVICE_INVALID will be returned, if an error occurs ST_FAILURE will be returned.

Remarks
ST_Find_Devices needs to have been run before a transducer can be opened.

ST_Close_Device
The **ST_Close_Device** function closes an open transducer, a transducer should always be closed before a program using the DLL exits.

```
ST_STATUS ST_Close_Device(
        DWORD device_id,
        BOOL force
        );
```

Parameters
device_id device id of the transducer to close, id's are indexed from 0 up to the number of devices found.
force reserved – set to FALSE.

Return value
If an open device is successfully closed, ST_SUCCESS will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if an error occurs ST_FAILURE will be returned.

ST_Close_ALL_Devices
The **ST_Close_ALL_Devices** function closes all open transducers, a transducer should always be closed before a program using the DLL exits.

```
ST_STATUS ST_Close_ALL_Devices(void);
```

Parameters
None

Return value
If all open devices are closed successfully, ST_SUCCESS will be returned, if an error occurs ST_FAILURE will be returned.

ST_GETINFO_Model
The **ST_GETINFO_Model** function returns the model name of the referenced transducer.

```
ST_STATUS ST_GETINFO_Model(
        DWORD device_id,
        char *model_string,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the model name of.
model_string	pointer to an array of chars, the maximum string length is 10.
bufsize	length of the array passed.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks
The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_SerialNumber
The **ST_GETINFO_SerialNumber** function returns the serial number in string form of the referenced transducer.

```
ST_STATUS ST_GETINFO_SerialNumber(
        DWORD device_id,
        char *serial_string,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the serial number of.
serial_string	pointer to an array of chars, the maximum string length is 9.
bufsize	length of the array passed.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks
The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_ID_String

The **ST_GETINFO_ID_String** function returns the ID string of the referenced transducer. The ID is a generic identifier which lists the model, serial and firmware revision.

```
ST_STATUS ST_GETINFO_ID_String(
        DWORD device_id,
        char *id_string,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the ID of.
id_string	pointer to an array of chars, the maximum string length is 59.
bufsize	length of the array passed.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks

The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ID string format:

[MODEL] - Firmware Revision: [FIRMWARE] Serial Number: [SERIAL]

Parameters have a fixed length:

MODEL – 9, FIRMWARE – 3, SERIAL – 8

Example: RWT321-DC - Firmware Revision: 3.6 Serial Number: 00000123

ST_GETINFO_Manufacture_Date

The **ST_GETINFO_Manufacture_Date** function returns the manufacture date of the referenced transducer. The date format used is DD/MM/YYYY.

```
ST_STATUS ST_GETINFO_Manufacture_Date(
        DWORD device_id,
        char *manufacture_date,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the manufacture date of.
manufacture_date	pointer to an array of chars, the maximum string length is 11.
bufsize	length of the array passed.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks

The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_Calibration_Date
The **ST_GETINFO_Calibration_Date** function returns the date that the referenced transducer was last calibrated. The date format used is DD/MM/YYYY.

```
ST_STATUS ST_GETINFO_Calibration_Date(
        DWORD device_id,
        char *calibration_date,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the last calibration date of.
calibration_date	pointer to an array of chars, the maximum string length is 11.
bufsize	length of the array passed.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks
The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_Customer
The **ST_GETINFO_Customer** function returns the registered customer name of the referenced transducer.

```
ST_STATUS ST_GETINFO_Customer(
        DWORD device_id,
        char *customer,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the registered customer name of.
customer	pointer to an array of chars, the maximum string length is 60.
bufsize	length of the array passed.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks
The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_ConnectionMethod

The **ST_GETINFO_ConnectionMethod** function returns the name of the interface that the referenced transducer is connected to. The interface is either COM## or USB.

```
ST_STATUS ST_GETINFO_ConnectionMethod(
        DWORD device_id,
        char *connection_method,
        DWORD bufsize
        );
```

Parameters

device_id	device id of the transducer to retrieve the connection method of.
connection_method	pointer to an array of chars, the maximum string length is 7.
bufsize	length of the array passed.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks

The returned string will be null terminated. The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_DeviceClass

The **ST_GETINFO_DeviceClass** function returns the device class of the referenced transducer. The device class identifies the type and underlying technology of the transducer.

```
ST_STATUS ST_GETINFO_DeviceClass(
        DWORD device_id,
        DWORD *DC
        );
```

Parameters

device_id — device id of the transducer to retrieve the technology class of.

DC — pointer to a variable of type DWORD that receives the device class index. The table below can be used to decode the index.

Index	Device Class
1	RWT Integrated – SAW Device
2	ORT Integrated – Optical Device
4	Strain Gauge Device
8	RWT External – SAW Device
16	ORT External – Optical Device

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, otherwise FT_FAILURE will be returned.

Remarks

The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_Firmware (Legacy)
The **ST_GETINFO_Firmware** function returns the firmware version (legacy format) of the referenced transducer.

```
ST_STATUS ST_GETINFO_Firmware(
        DWORD device_id,
        float *firmware
        );
```

Parameters

device_id	device id of the transducer to retrieve the firmware version of.
firmware	pointer to a variable of type float that receives the firmware version.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, otherwise FT_FAILURE will be returned.

Remarks
The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

This command is considered legacy, the float variable does not accurately convey the firmware revision.

ST_GETINFO_FirmwareEx
The **ST_GETINFO_FirmwareEx** function returns the firmware revision and build of the referenced transducer.

```
ST_STATUS ST_GETINFO_FirmwareEx(
        DWORD device_id,
        VERSIONS *verinfo
        );
```

Parameters

device_id	device id of the transducer to retrieve the firmware version of.
verinfo	pointer to a variable of type VERSIONS that receives the firmware information.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, otherwise FT_FAILURE will be returned.

Remarks
The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

For firmware earlier than 5, the revision is converted from a simpler M.m format and the build is 1.

ST_GETINFO_FirmwareText

The **ST_GETINFO_FirmwareText** function decodes the firmware revision and build of the referenced transducer into a text string.

```
ST_STATUS ST_GETINFO_FirmwareText(
        DWORD device_id,
        char *firmwaretxt,
        DWORD bufsize);
```

Parameters

device_id	device id of the transducer to retrieve the firmware version of.
firmwaretxt	pointer to an array of chars to receive the firmware string. The maximum string length is 60.
bufsize	length of the array passed.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid ST_DEVICE_INVALID will be returned, if the buffer passed to the function is too small ST_BUFFER_TOO_SMALL will be returned, otherwise FT_FAILURE will be returned.

Remarks

The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

For firmware earlier than 5, the revision is converted from a simpler M.m format and the build is 1.

Firmware Text Example: 5.2.1 Build 12345

ST_GETINFO_FSD

The **ST_GETINFO_FSD** function returns the full scale rating of the referenced transducer.

```
ST_STATUS ST_GETINFO_FSD(
        DWORD device_id,
        DWORD *fsd
        );
```

Parameters

device_id	device id of the transducer to retrieve the FSD of.
fsd	pointer to a variable of type DWORD that receives the full scale rating.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, otherwise FT_FAILURE will be returned.

Remarks

The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

Use the **ST_GETINFO_Units** function to get the transducers native unit of measurement for torque.

ST_GETINFO_Units

The **ST_GETINFO_Units** function returns the native measurement unit of the referenced transducer. The FSD and all torque values use this unit as the unit of measurement.

```
ST_STATUS ST_GETINFO_Units(
        DWORD device_id,
        DWORD *units
        );
```

Parameters

device_id — device id of the transducer to retrieve the measurement unit of.

units — pointer to a variable of type DWORD that receives the unit index. The table below can be used to decode the index.

Index	Unit
0	ozf.in
1	lbf.in
2	lbf.ft
3	gf.cm
4	Kgf.cm
5	Kgf.m
6	mN.m
7	N.m

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, otherwise FT_FAILURE will be returned.

Remarks

The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

ST_GETINFO_Max_Speed
The **ST_GETINFO_Max_Speed** function returns the maximum speed scaling for the referenced transducer.

```
ST_STATUS ST_GETINFO_Max_Speed(
        DWORD device_id,
        DWORD *maxspeed
        );
```

Parameters

device_id	device id of the transducer to retrieve the maximum speed of.
maxspeed	pointer to a variable of type DWORD that receives the maximum speed scaling value.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if speed isn't fitted ST_SPEED_NOT_FITTED will be returned, otherwise FT_FAILURE will be returned.

Remarks
The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

The speed value does not limit the transducers ability to measure higher speeds, but is the maximum speed that is specified by the customer. The max speed value is the maximum value that the analog output can be scaled to.

ST_GETINFO_Speed_Gratings
The **ST_GETINFO_Speed_Gratings** function returns the number of slots in the speed disk grating.

```
ST_STATUS ST_GETINFO_Speed_Gratings(
        DWORD device_id,
        DWORD *gratings
        );
```

Parameters

device_id	device id of the transducer to retrieve the grating size of.
gratings	pointer to a variable of type DWORD that receives the number of slots in the speed disk grating.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if speed isn't fitted ST_SPEED_NOT_FITTED will be returned, otherwise FT_FAILURE will be returned.

Remarks
The device does not need to be open to use this function and it can be called during the find device process, provided that the waitforcomplete parameter is FALSE and the device id is less than the number of devices found.

The speed disk grating is used to measure speed, the number of slots in the grating sets the measurement resolution.

ST_GET_Data_Block

The **ST_GET_Data_Block** function returns a transducer data set from the referenced transducer. The data set contains all the data that can be captured from the transducer.

```
ST_STATUS ST_GET_Data_Block(
        DWORD device_id,
        ST_DATABLOCK *dat
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type ST_DATABLOCK that receives the complete transducer data set.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. The torque value will be in the native unit of measurement for the transducer. The ST_DATABLOCK custom variable type is defined in the DLL Structures section of this document. The peak values can be manually reset using the **ST_RESET_Peaks** function.

ST_GET_Data_Block_Extract

The **ST_GET_Data_Block_Extract** function returns the individual components of the transducer data set from the referenced transducer. The data set contains all the data that can be captured from the transducer.

```
ST_STATUS ST_GET_Data_Block_Extract(
        DWORD device_id,
        float *torque,
        float *ptorque,
        float *artorque,
        float *cwptorque,
        float *ccwptorque,
        float *mintorque,
        float *maxtorque,
        DWORD *speedfast,
        DWORD *pspeedfast,
        DWORD *speedslow,
        DWORD *pspeedslow,
        float *powerwattsfast,
        float *powerwattsslow,
        float *powerhpfast,
        float *powerhpslow,
        float *ppowerwattsfast,
        float *ppowerwattsslow,
        float *ppowerhpfast,
        float *ppowerhpslow,
        float *tmpambient,
        float *tmpshaft,
        DWORD *ticktock
        );
```

Parameters

device_id	device id of the transducer to retrieve the data from.
torque	pointer to a variable of type float that returns the current torque value.
ptorque	pointer to a variable of type float that returns the peak torque value.
artorque	pointer to a variable of type float that returns the auto reset torque value.
cwptorque	pointer to a variable of type float that returns the clockwise peak torque value.
ccwptorque	pointer to a variable of type float that returns the counter-clockwise peak torque value.
mintorque	pointer to a variable of type float that returns the minimum torque value from the reference/reset point.
maxtorque	pointer to a variable of type float that returns the maximum torque value from the reference/reset point
speedfast	pointer to a variable of type DWORD that returns the current speed value from the fast capture mode.
pspeedfast	pointer to a variable of type DWORD that returns the peak speed value captured from the fast capture mode.
speedslow	pointer to a variable of type DWORD that returns the current speed value from the slow capture mode.

pspeedslow	pointer to a variable of type DWORD that returns the peak speed value captured from the slow capture mode.
powerwattsfast	pointer to a variable of type float that returns the current power value in watts, based on the speed from the fast capture mode.
powerwattsslow	pointer to a variable of type float that returns the current power value in watts, based on the speed from the slow capture mode.
powerhpfast	pointer to a variable of type float that returns the current power value in horse power, based on the speed from the fast capture mode.
powerhpslow	pointer to a variable of type float that returns the current power value in horse power, based on the speed from the slow capture mode.
ppowerwattsfast	pointer to a variable of type float that returns the peak power value in watts, based on the speed from the fast capture mode.
ppowerwattsslow	pointer to a variable of type float that returns the peak power value in watts, based on the speed from the slow capture mode.
ppowerhpfast	pointer to a variable of type float that returns the peak power value in horse power, based on the speed from the fast capture mode.
ppowerhpslow	pointer to a variable of type float that returns the peak power value in horse power, based on the speed from the slow capture mode.
tmpambient	pointer to a variable of type float that returns the transducers ambient temperature (degrees Celsius).
tmpshaft	pointer to a variable of type float that returns the transducers shaft temperature (degrees Celsius).
ticktock	pointer to a variable of type DWORD that returns the elapsed time in milliseconds from the time stamp start/reset point.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. The torque value will be in the native unit of measurement for the transducer. The ticktock parameter is part of a time stamp system added to aid LabView/Torqview accurately time stamp readings, refer to the **ST_GET_TimeStamp** and **ST_Reset_TimeStamp** function descriptions and the Time Stamp section of this document. The peak values can be manually reset using the **ST_RESET_Peaks** function.

The **ST_GET_Data_Block_Extract** function differs from the **ST_GET_Data_Block** function, one returns the components in a single block as a single parameter, while the other returns the components individually as separate parameters. The **ST_GET_Data_Block_Extract** function can be used with languages that cannot create custom types.

ST_GET_Torque_Select
The **ST_GET_Torque_Select** function selects a specified torque type and returns its value from the referenced transducer.

```
ST_STATUS ST_GET_Torque_Select (
        DWORD device_id,
        DWORD torqueselect,
        float *dat
        );
```

Parameters

device_id — device id of the transducer to retrieve data from.

torqueselect — selects the torque value to be returned in dat. The table below shows the parameter values for the different torque types.

Value	Torque Type
0	Current torque
1	Peak torque
2	Peak torque with auto reset
3	Peak clockwise torque
4	Peak counter-clockwise torque
5	Maximum torque from reference/reset
6	Minimum torque from reference/reset
7	Maximum/Minimum torque from reference/reset (MINMAX_TMP – see remarks)

dat — pointer to a variable of type float that returns the torque value selected by torqueselect.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. If selecting torque type 7, you will need to pass a MINMAX_TMP structure and cast the pointer to a float. The torque value will be in the native unit of measurement for the transducer. The peak torque values can be manually reset using the **ST_RESET_Peaks** function.

ST_GET_Torque_Select_Convert

The **ST_GET_Torque_Select_Convert** function selects a specified torque type, converts the torque value into the unit of measurement specified and returns the converted value for the referenced transducer.

```
ST_STATUS ST_GET_Torque_Select_Convert (
        DWORD device_id,
        DWORD torqueselect,
        DWORD convertto,
        float *dat
        );
```

Parameters

device_id — device id of the transducer to retrieve data from.

torqueselect — selects the torque value to be converted and returned in dat. The table below shows the parameter values for the different torque types.

Value	Torque Type
0	Current torque
1	Peak torque
2	Peak torque with auto reset
3	Peak clockwise torque
4	Peak counter-clockwise torque
5	Maximum torque from reference/reset
6	Minimum torque from reference/reset
7	Maximum/Minimum torque from reference/reset (MINMAX_TMP – see remarks)

convertto — selects the unit of measurement that the selected torque value will be converted to. The table below shows the parameter values for the different units of measurement.

Value	Unit
0	ozf.in
1	lbf.in
2	lbf.ft
3	gf.cm
4	Kgf.cm
5	Kgf.m
6	mN.m
7	N.m

dat pointer to a variable of type float that returns the converted torque value selected by torqueselect.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. If selecting torque type 7, you will need to pass a MINMAX_TMP structure and cast the pointer to a float. The peak torque values can be manually reset using the **ST_RESET_Peaks** function.

ST_GET_Torque

The **ST_GET_Torque** function returns the current torque value for the referenced transducer.

```
ST_STATUS ST_GET_Torque(
        DWORD device_id,
        float *dat
        );
```

Parameters

device_id device id of the transducer to retrieve data from.

dat pointer to a variable of type float that receives the current torque value.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. The torque value will be in the native unit of measurement for the transducer.

ST_GET_Torque_Peak
The **ST_GET_Torque_Peak** function returns the peak torque value for the referenced transducer.

```
ST_STATUS ST_GET_Torque_Peak(
        DWORD device_id,
        float *dat
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type float that receives the peak torque value.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. The torque value will be in the native unit of measurement of the transducer. The peak torque value can be manually reset using the **ST_RESET_Peaks** function.

ST_GET_Torque_Auto_Reset
The **ST_GET_Torque_Auto_Reset** function returns the peak torque with auto reset value for the referenced transducer.

```
ST_STATUS ST_GET_Torque_Auto_Reset(
        DWORD device_id,
        float *dat
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type float that receives the peak torque with auto reset value.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. The torque value will be in the native unit of measurement of the transducer. The Peak Torque with auto reset value can be manually reset using the **ST_RESET_Peaks** function.

ST_GET_Torque_Peak_MinMax
The **ST_GET_Torque_Peak_MinMax** function returns the maximum and minimum torque value from a reference/reset point for the referenced transducer.

```
ST_STATUS ST_GET_Torque_Peak_MinMax(
        DWORD device_id,
        MINMAX_TMP *dat,
        BOOL reset_minmax
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type MINMAX_TMP that receives the minimum/maximum torque values.
reset_minmax	if TRUE the reference point for the minimum/maximum torque values is reset after value retrieval. If FALSE the minimum/maximum torque values are not reset.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. The torque value will be in the native unit of measurement of the transducer. The reference point for the minimum/maximum values can be reset using the reset_minmax parameter or by using the **ST_RESET_Peaks** function.

ST_GET_Speed_Fast
The **ST_GET_Speed_Fast** function returns the current fast mode speed value for the referenced transducer.

```
ST_STATUS ST_GET_Speed_Fast(
        DWORD device_id,
        DWORD *dat
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type DWORD that receives the current fast mode speed value.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. Refer to the Speed Modes section for a definition of the different speed capture modes.

ST_GET_Speed_Slow
The **ST_GET_Speed_Slow** function returns the current slow mode speed value for the referenced transducer.

```
ST_STATUS ST_GET_Speed_Slow(
        DWORD device_id,
        DWORD *dat
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type DWORD that receives the current slow mode speed value.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. Refer to the Speed Modes section for a definition of the different speed capture modes.

ST_GET_Power_In_Watts

The **ST_GET_Power_In_Watts** function returns the current power value in watts derived from the current torque and speed values for the referenced transducer.

```
ST_STATUS ST_GET_Power_In_Watts(
        DWORD device_id,
        float *dat,
        DWORD speed_mode
        );
```

Parameters

device_id — device id of the transducer to retrieve data from.

dat — pointer to a variable of type float that receives the current power value in watts.

speed_mode — selects the speed capture mode that the power value should be calculated from. The table below shows the parameter values for the different speed modes

Value	Speed Modes
0	Slow
1	Fast

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. Refer to the Speed Modes section for a definition of the different speed capture modes.

ST_GET_Power_In_HP

The **ST_GET_Power_In_HP** function returns the current power value in horse power derived from the current torque and speed values for the referenced transducer.

```
ST_STATUS ST_GET_Power_In_HP(
        DWORD device_id,
        float *dat,
        DWORD speed_mode
        );
```

Parameters

device_id — device id of the transducer to retrieve data from.

dat — pointer to a variable of type float that receives the current power value in horse power.

speed_mode — selects the speed capture mode that the power value should be calculated from. The table below shows the parameter values for the different speed modes

Value	Speed Modes
0	Slow
1	Fast

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before using this function. Refer to the Speed Modes section for a definition of the different speed capture modes.

ST_GET_Temperature_Ambient
The **ST_GET_Temperature_Ambient** function returns the measured ambient temperature for the referenced transducer.

ST_STATUS **ST_GET_Temperature_Ambient**(
DWORD device_id,
float *dat
);

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type float that receives the ambient temperature in degrees Celsius.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. Refer to the Temperature Sensors section for more information.

ST_GET_Temperature_Shaft
The **ST_GET_Temperature_Shaft** function returns the measured shaft temperature for the referenced transducer.

ST_STATUS **ST_GET_Temperature_Shaft**(
DWORD device_id,
float *dat
);

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type float that receives the shaft temperature in degrees Celsius.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. Refer to the Temperature Sensors section for more information.

ST_GET_Temperature_Internal
The **ST_GET_Temperature_Internal** function returns the measured internal temperature for the referenced transducer.

ST_STATUS **ST_GET_Temperature_Internal**(
DWORD device_id,
float *dat
);

Parameters

device_id	device id of the transducer to retrieve data from.
dat	pointer to a variable of type float that receives the internal temperature in degrees Celsius.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. Refer to the Temperature Sensors section for more information.

ST_GET_Torque_Filter
The **ST_GET_Torque_Filter** function returns the current torque filter setting.

ST_STATUS **ST_GET_Torque_Filter**(
DWORD device_id,
DWORD *filtervalue
);

Parameters

device_id	device id of the transducer to retrieve data from.
filtervalue	pointer to a variable of type DWORD that receives the current torque filter setting. A filter value of 0 equals OFF.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function.

ST_SET_Torque_Filter
The **ST_SET_Torque_Filter** function configures the torque filter.

```
ST_STATUS ST_SET_Torque_Filter(
        DWORD device_id,
        DWORD filtervalue,
        BOOL save
        );
```

Parameters

device_id	device id of the transducer to configure.
filtervalue	configures the torque filter setting, a value greater than zero sets the number of samples used in the filter; valid values are 0(OFF), 2, 4, 8, 16, 32, 64, 128, and 256.
save	if TRUE the specified filter will be enabled, saved and retained across power cycles of the transducer. If FALSE the specified setting will be enabled, but on reset the transducer will revert back to the default or previously saved setting.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function.

ST_GET_Speed_Filter
The **ST_GET_Speed _Filter** function returns the current speed filter setting.

```
ST_STATUS ST_GET_Speed _Filter(
        DWORD device_id,
        DWORD *filtervalue
        );
```

Parameters

device_id	device id of the transducer to retrieve data from.
filtervalue	pointer to a variable of type DWORD that receives the current speed filter setting. A filter value of 0 equals OFF.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, if speed isn't fitted ST_SPEED_NOT_FITTED will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function.

ST_SET_Speed _Filter
The **ST_SET_Speed _Filter** function configures the speed filter.

```
ST_STATUS ST_SET_Speed _Filter(
        DWORD device_id,
        DWORD filtervalue,
        BOOL save
        );
```

Parameters

device_id	device id of the transducer to configure.
filtervalue	configures the speed filter setting, a value greater than zero sets the number of samples used in the filter; valid values are 0(OFF), 2, 4, 8, 16, 32, 64, 128, and 256.
save	if TRUE the specified filter will be enabled, saved and retained across power cycles of the transducer. If FALSE the specified setting will be enabled, but on reset the transducer will revert back to the default or previously saved setting.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, if speed isn't fitted ST_SPEED_NOT_FITTED will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function.

ST_RESET_Peaks
The **ST_RESET_Peaks** function resets the stored torque, speed and power peak values as selected by the specified flags.

```
ST_STATUS ST_RESET_Peaks(
        DWORD device_id,
        DWORD reset_flags
        );
```

Parameters

device_id — device id of the transducer to access.

reset_flags — selects the stored peak value(s) to be reset. Peak values are selected by passing the flag value of the peak to be reset. Multiple peak values can be reset at the same time by combining flags, this is done by adding or OR'ing the required flag values together.

Flag Value	Value to be reset	Description
0x4	Peak torque	Resets the peak torque to zero.
0x8	Peak torque with auto reset	Resets the peak torque with auto reset to zero.
0x10	Peak torque CW	Resets the peak torque clockwise to zero.
0x20	Peak torque CCW	Resets the peak torque counter-clockwise to zero.
0x40	Peak min max	Resets the min and max values to the current torque value.
0x80	Peak fast mode speed	Resets the peak fast mode speed to zero.
0x100	Peak slow mode speed	Resets the peak slow mode speed to zero.
0x200	Peak fast mode power	Resets the peak power using the fast speed mode to zero.
0x400	Peak slow mode power	Resets the peak power using the slow speed mode to zero.
0x800	Angle (rotations & degrees)	Resets the rotations and degrees to zero.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. Reset all values by sending 0xFFC as the value for reset_flags.

ST_Zero_Transducer
The **ST_Zero_Transducer** function zeros the tranducers torque value, this is done by recording the current torque value as an offset, the offset is then subtracted from all subsequant torque readings.

ST_STATUS **ST_Zero_Transducer**(
DWORD device_id
);

Parameters
device_id device id of the transducer to zero.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. The zero offset is lost when the transducer is power cycled.

ST_ZeroAverage_Transducer
The **ST_ZeroAverage_Transducer** function zeros the tranducers torque value using a 32 sample average. The current torque value is sampled 32 times and averaged, the averaged value is then recorded as an offset and is subtracted from all subsequant torque readings.

ST_STATUS **ST_ZeroAverage_Transducer**(
DWORD device_id
);

Parameters
device_id device id of the transducer to zero.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. The zero offset is lost when the transducer is power cycled.

ST_Reset_TimeStamp
The **ST_Reset_TimeStamp** function initialises and resets the time stamp counter. Calls to **ST_GET_TimeStamp** function will return elapsed time from this reset point.

void **ST_Reset_TimeStamp**(void);

Parameters
None

Return value
None.

Remarks
For more information on the time stamp functionality refer to the Time Stamp section of this document

ST_GET_TimeStamp
The **ST_GET_TimeStamp** function returns the elapsed time from start/reset point.

DWORD **ST_GET_TimeStamp** (void);

Parameters
None

Return value
Elapsed time in milliseconds from the start/reset point.

Remarks
If the TimeStamp counter has not been initialised using the **ST_Reset_TimeStamp** function, calls to this function will always return zero. For more information on the time stamp functionality refer to the Time Stamp section of this document.

ST_Capture_Enable
The **ST_Capture_Enable** function initialises and starts the automated capture mode. Once started data is captured and buffered at the rate requested. Data is extracted from the buffer by using **ST_GET_Capture_Data** and the capture is stopped by using **ST_Capture_Disable.**

```
ST_STATUS ST_Capture_Enable (
        DWORD device_id,
        DWORD caprate
        );
```

Parameters

device_id	device id of the transducer to configure.
caprate	the data capture rate to be used by the capture mode. The value should be specified as the number of captures per second.

Return value
If the function completes successfully ST_SUCCESS will be returned and the capture proccess will have been started. If the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open before using this function. The caprate parameter must be between 1 and the maximum supported capture rate. Use the **ST_GET_Capture_Rate** function to retieve the maximum capture rate.

The requested capture rate may not be possible due to the resolution of the capture timer. The DLL divides the timer value for a second by the capture rate, and uses that value as the timer value. The operating system timers must be capable of a 1ms resolution for the capture system to work.

ST_Capture_Disable
The **ST_Capture_Disable** function stops an active data capture mode and frees all allocated resources. Captured data which has not been extracted will be purged.

```
ST_STATUS ST_Capture_Disable (
        DWORD device_id
        );
```

Parameters
device_id device id of the transducer to configure.

Return value
If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, otherwise FT_FAILURE will be returned.

Remarks
The referenced device needs to be open and the capture mode active before this function can be used.

ST_GET_Capture_Data

The **ST_GET_Capture_Data** function extracts data from an active data capture mode. Data is stored and transferred as record blocks, the **ST_GET_Capture_Data** function transfers these record blocks from an internal ring buffer to the users record array.

```
ST_STATUS ST_GET_Capture_Data (
        DWORD device_id,
        CAPREC *record_ptr,
        DWORD records,
        DWORD *recordno
        );
```

Parameters

device_id	device id of the transducer where there is an active data capture mode.
record_ptr	pointer to an array of CAPREC records.
records	number of records in the array pointed to by record_ptr.
recordno	pointer to a variable of type DWORD that receives the number of records written to the array pointed to by record_ptr.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, if there is no active data capture mode ST_CMD_INACTIVE will be returned.

ST_BUFFER_OVERFLOW will be returned if the primary and secondary internal buffers become fully consumed. This would occur if the **ST_GET_Capture_Data** poll rate is too low compared to the selected capture rate. The capture mode will be terminated if this occurs.

ST_NO_COMMS_IN_PROCESS will be returned if the data feed from the transducer stops. The capture mode will be terminated if this occurs.

FT_FAILURE will be returned for all other errors occurs.

Remarks

The referenced device needs to be open and the capture mode active before this function can be used. The size of the array passed to **ST_GET_Capture_Data** should be carefully considered and take in to acount the configured capture rate and time between calls to **ST_GET_Capture_Data**.

ST_GET_Capture_Rate

The **ST_GET_Capture_Rate** function retrieves the maximum capture rate that can be used with the data capture mode.

```
ST_STATUS ST_GET_Capture_Rate(
        DWORD device_id,
        DWORD *caprate
        );
```

Parameters

device_id	device id of the transducer to access.
caprate	pointer to a variable of type DWORD that receives the maximum data capture rate supported by the transducer/DLL.

Return value

If the function completes successfully ST_SUCCESS will be returned, if the device_id is invalid then ST_DEVICE_INVALID will be returned, if the device is not open ST_DEVICE_NOT_OPEN will be returned, if the operating system does not support a 1ms timer resolution then ST_WINDOWS_TIMER_RESOLUTION will be returned, otherwise FT_FAILURE will be returned.

Remarks

The referenced device needs to be open before this function can be used. The operating system timers must be capable of a 1ms resolution for the capture system to work.