Touch Sensing Software

API Reference Manual

TSSAPIRM
Rev. 8
08/2013

freescale™

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: TSSAPIRM
Rev. 8, 08/2013

freescale™

Chapter 1

Touch Sensing Software Library Overview

1.1 Introduction 1-1
1.2 Library features 1-1
1.3 Library architecture 1-2
1.4 System base modules 1-3
1.4.1 System setup module 1-3
1.4.2 GPIO module 1-4
1.4.3 Hardware timer module 1-4
1.5 Signal sensing modules 1-5
1.5.1 GPIO low level sensor method 1-5
1.5.2 TSI low level sensor method 1-5
1.5.3 TSI Lite low level sensor method 1-6
1.6 Key detector modules 1-7
1.6.1 Basic key detector module 1-7
1.6.2 AFID key detector module 1-8
1.6.3 Noise key detector module 1-9
1.7 Decoder module 1-9
1.8 System configuration and management module 1-10

Chapter 2

TSS System setup parameters

Chapter 3

Application Interface

3.1 TSS initialization function 3-1
3.2 TSS task function 3-1
3.3 TSS task sequenced function 3-2
3.4 TSS Library Configuration Save and Restore Functions 3-3
3.4.1 Save and restore electrode data 3-3
3.4.2 Save and restore module data 3-4
3.4.3 Save and restore all system data 3-6
3.5 TSS Library Configuration and Status Registers 3-7
3.5.1 Writing to the Configuration and Status Registers 3-7
3.5.2 Reading the Configuration and Status registers 3-8
3.5.3 Configuration and Status registers list 3-10
3.5.4 Faults register 3-11
3.5.5 System Configuration register 3-13
3.5.6 Number of Samples register 3-15
3.5.7 DC Tracker Rate register 3-15

3.5.8 Slow DC Tracker Factor register 3-16
3.5.9 Response Time register 3-16
3.5.10 Stuck-key Timeout register 3-16
3.5.11 Low Power Electrode register 3-17
3.5.12 Low-Power Electrode Sensitivity register 3-18
3.5.13 System Trigger register 3-18
3.5.14 Sensitivity Configuration register 3-19
3.5.15 Electrode enablers 3-19
3.5.16 Electrode status 3-20
3.5.17 Baseline register 3-20
3.5.18 Dc-Tracker enablers 3-21
3.5.19 Configuration Checksum Register 3-21
3.6 Keypad decoder API 3-22
3.6.1 Writing to the Configuration and Status registers 3-22
3.6.2 Reading the Configuration and Status registers 3-23
3.6.3 Configuration and Status registers list 3-25
3.6.4 Control ID register 3-26
3.6.5 Control Configuration register 3-26
3.6.6 Buffer Pointer register 3-27
3.6.7 BufferReadIndex 3-28
3.6.8 BufferWriteIndex 3-28
3.6.9 Event Control and Status register 3-29
3.6.10 MaxTouches register 3-30
3.6.11 Auto Repeat Rate register 3-31
3.6.12 Auto Repeat Start register 3-31
3.6.13 Keypad Callback function 3-32
3.7 Slider and Rotary decoder API 3-32
3.7.1 Writing to the Configuration and Status registers 3-33
3.7.2 Reading the Configuration and Status registers 3-34
3.7.3 Configuration and Status registers list 3-36
3.7.4 Control ID register 3-36
3.7.5 Control configuration 3-37
3.7.6 Dynamic Status register 3-37
3.7.7 Static Status register 3-38
3.7.8 Events Control register 3-39
3.7.9 Auto-repeat Rate register 3-40
3.7.10 Movement Timeout register 3-40
3.7.11 Callback function 3-41
3.8 Analog slider and analog rotary decoder API 3-41
3.8.1 Writing to the Configuration and Status registers 3-42
3.8.2 Reading the Configuration and Status registers 3-43
3.8.3 Configuration and Status registers list 3-45
3.8.4 Control ID register 3-46
3.8.5 Control configuration 3-46
3.8.6 Dynamic Status register 3-47

3.8.7 Position register 3-48
3.8.8 Events Control register 3-48
3.8.9 Auto-repeat Rate register 3-49
3.8.10 Movement Timeout register 3-50
3.8.11 Range register 3-50
3.8.12 Callback function 3-51
3.9 Matrix decoder API 3-51
3.9.1 Writing to the Configuration and Status registers 3-51
3.9.2 Reading the Configuration and Status registers 3-52
3.9.3 Configuration and Status registers list 3-54
3.9.4 Control ID register 3-55
3.9.5 Control configuration 3-56
3.9.6 Events Control register 3-56
3.9.7 Auto-repeat Rate register 3-58
3.9.8 Movement Timeout register 3-58
3.9.9 Dynamic Status X register 3-59
3.9.10 Dynamic Status Y register 3-59
3.9.11 Position X register 3-60
3.9.12 Position Y register 3-60
3.9.13 Gesture distance X register 3-61
3.9.14 Gesture distance Y register 3-61
3.9.15 Range X register 3-62
3.9.16 Range Y register 3-62
3.9.17 Matrix callback function 3-63

Chapter 4

Library Intermediate Layer Interfaces

4.1 Capacitive sensing and key detector interface 4-1
4.1.1 Electrode sampling 4-1
4.1.2 Low-level initialization 4-2
4.2 Decoder interface 4-4
4.2.1 Decoder main function 4-4
4.2.2 Writing the decoder schedule counter 4-5
4.2.3 Reading the decoder schedule counter 4-5
4.2.4 Reading the instant delta in the control 4-6

Chapter 5

C++ wrapper

5.1 TSSSystem class 5-1
5.1.1 TSSTask 5-1
5.1.2 isElecTouched 5-2
5.1.3 isElecEnabled 5-2

5.1.4 enableElec . . . 5-3
5.1.5 disableElec . . . 5-3
5.1.6 setSensitivity . . . 5-3
5.1.7 getSensitivity . . . 5-4
5.1.8 enableDCTracker . . . 5-4
5.1.9 disableDCTracker . . . 5-5
5.1.10 setBaseline . . . 5-5
5.1.11 getBaseline . . . 5-6
5.1.12 enable . . . 5-6
5.1.13 disable . . . 5-6
5.1.14 set . . . 5-7
5.1.15 get . . . 5-7
5.1.16 callbackSysFaults . . . 5-8
5.1.17 onFault . . . 5-8
5.1.18 regCallback . . . 5-9
5.1.19 unregCallback . . . 5-9
5.2 TSSControlFactory class . . . 5-9
5.2.1 createTSSControl . . . 5-9
5.2.2 getTSSControl . . . 5-10
5.3 TSSControl class . . . 5-10
5.3.1 callbackControl . . . 5-10
5.3.2 enable . . . 5-11
5.3.3 disable . . . 5-11
5.3.4 set . . . 5-12
5.3.5 get . . . 5-12
5.4 TSSKeypad class . . . 5-13
5.4.1 callbackControl . . . 5-13
5.4.2 enable . . . 5-13
5.4.3 disable . . . 5-14
5.4.4 set . . . 5-14
5.4.5 get . . . 5-15
5.4.6 getControlStruct . . . 5-15
5.4.7 regCallback . . . 5-15
5.4.8 unregCallback . . . 5-16
5.4.9 onTouch . . . 5-16
5.4.10 onRelease . . . 5-17
5.4.11 onExceededKeys . . . 5-17
5.5 TSSASlider class . . . 5-17
5.5.1 callbackControl . . . 5-17
5.5.2 enable . . . 5-18
5.5.3 disable . . . 5-18
5.5.4 set . . . 5-19
5.5.5 get . . . 5-19
5.5.6 getControlStruct . . . 5-20
5.5.7 regCallback . . . 5-20

5.5.8 unregCallback 5-20
5.5.9 onInitialTouch 5-21
5.5.10 onAllRelease 5-21
5.5.11 onMovement 5-21
5.6 TSSARotary class 5-22
5.6.1 callbackControl 5-22
5.6.2 enable 5-22
5.6.3 disable 5-23
5.6.4 set 5-23
5.6.5 get 5-24
5.6.6 getControlStruct 5-24
5.6.7 regCallback 5-24
5.6.8 unregCallback 5-25
5.6.9 onInitialTouch 5-25
5.6.10 onAllRelease 5-26
5.6.11 onMovement 5-26
5.7 TSSSlider class 5-26
5.7.1 callbackControl 5-26
5.7.2 enable 5-27
5.7.3 disable 5-27
5.7.4 set 5-28
5.7.5 get 5-28
5.7.6 getControlStruct 5-29
5.7.7 regCallback 5-29
5.7.8 unregCallback 5-30
5.7.9 onInitialTouch 5-30
5.7.10 onAllRelease 5-30
5.7.11 onMovement 5-31
5.8 TSSRotary class 5-31
5.8.1 callbackControl 5-31
5.8.2 enable 5-31
5.8.3 disable 5-32
5.8.4 set 5-32
5.8.5 get 5-33
5.8.6 getControlStruct 5-33
5.8.7 regCallback 5-34
5.8.8 unregCallback 5-34
5.8.9 onInitialTouch 5-35
5.8.10 onAllRelease 5-35
5.8.11 onMovement 5-35
5.9 TSSMatrix class 5-36
5.9.1 callbackControl 5-36
5.9.2 enable 5-36
5.9.3 disable 5-36
5.9.4 set 5-37

5.9.5 get 5-37
5.9.6 getControlStruct 5-38
5.9.7 regCallback 5-38
5.9.8 unregCallback 5-39
5.9.9 onInitialTouch 5-39
5.9.10 onAllRelease 5-40
5.9.11 onMovement 5-40
5.9.12 onGestures 5-40

Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web are the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, see freescale.com.

The following revision history table summarizes changes in this document.

Revision Number	Revision Date	Description of Changes
Rev. 1	07/2009	Launch Release for TSS 0.2.1
Rev. 2	10/2009	Updated for TSS 1.0 release
Rev. 3	11/2009	Updated for TSS 1.1 release
Rev. 4	06/2010	Updated for TSS 2.0 release
Rev. 5	04/2011	Updated for TSS 2.5 release
Rev. 6	03/2012	Updated for TSS 2.6 release
Rev. 7	08/2012	Updated for TSS 3.0 release Added a new text with the sections “Signal normalization”,“Automatic sensitivity calibration”,”Baseline initialization”,”Electrodes groups”,“Analog slider and analog rotary decoder API”,“Reading the instant delta in the control”,”Proximity function”,”Automatic Sensitivity Calibration”,”Shielding function and Water tolerance”,”Water tolerance mode”,”Analog rotary”,”Analog Slider”,”Matrix”
Rev. 8	08/2013	Updated for TSS 3.1 release Added a new text with the sections “Key detector modules”, “Basic key detector module”,“AFID key detector module”,“Noise key detector module”,“TSS Library Configuration Save and Restore Functions”,“Save and restore electrode data”,“Save and restore module data”,“Save and restore all system data”,“Slow DC Tracker Factor register”,“Baseline register”,“Dc-Tracker enablers”, Removed sections 2.2, 2.3, 3.4.10, 3.4.14, 4.2.4, Appendix

Chapter 1
Touch Sensing Software Library Overview

This chapter describes features, architecture, and software modules of the Touch Sensing Software (TSS) library. The following chapters describe library API functions and the use of resources.

1.1 Introduction

The TSS library enables capacitive sensing for all Freescale S08 and ColdFire V1, ColdFire+, Kinetis Microcontroller families based on ARM®Cortex™-M4 and ARM®Cortex™-M0+ cores, providing the common touch sense decoding structures such as keypad, rotary, slider, analog slider, analog rotary, and matrix. It is implemented in a software-layered architecture to enable easy integration into the application code and migration to other Freescale MCUs and customer customization.

1.2 Library features

The TSS library features include:

- Configurable number of electrodes from 1 to 64
- Configurable number of keypads, rotaries, sliders, analog sliders, analog rotaries, and matrixes
- Automatic electrode sensitivity calibration
- Three key detector methods (Basic, AFID and Noise)
- False detection prevention against external environment
- Electrode fault detection
- IIR and noise amplitude filters
- Shielding function
- Support for water-resistant touch systems
- Use of any standard MCU I/O as an electrode
- Touch Sense Input (TSI) module support
- TSI noise mode support
- One or more timer modules used with GPIO sampling algorithm
- Capability to wake from the MCU’s low power mode via the TSI module
- Three triggering modes (ALWAYS, SW, and AUTO when the TSI module is used)
- MISRA-compliant library encoding
- Easy application integration achieved by a modular software layer architecture

1.3 Library architecture

The TSS library is implemented by using modular software layering shown in the Figure 1-1. It provides a framework for Freescale touch sensing solutions and allows further integration of new modules for application-specific needs. It is structured only for linking the user application with the modules used in the application code. Additionally, it supports module use for any layer in the user application library. Section 1.4, “System base modules” describes each module of the library.

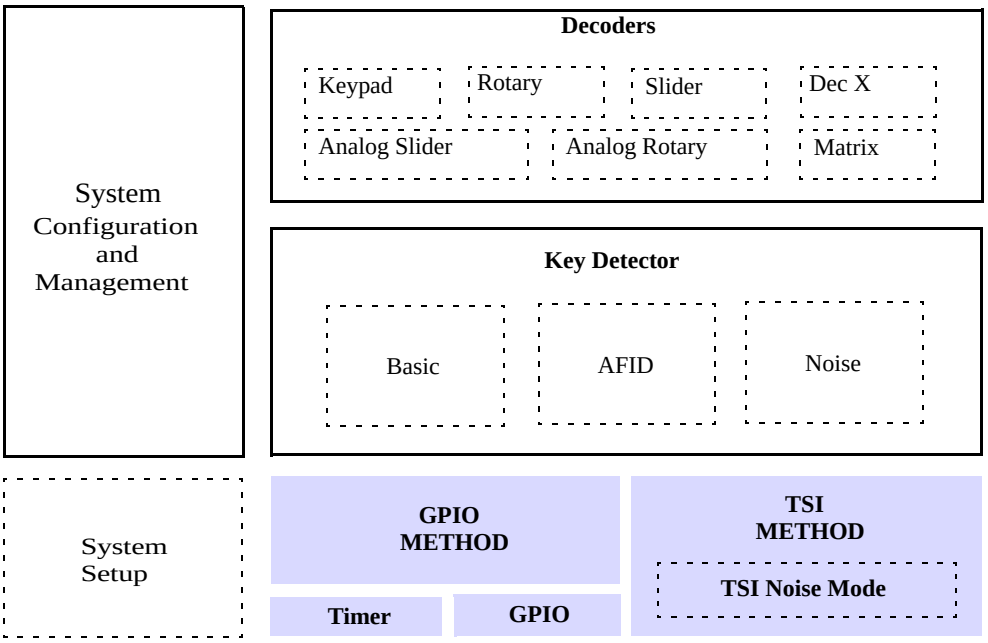

Figure 1-1. TSS library architecture

Figure 1-2 is an example of the application project using the ARM Cortex-M4 version of library files. As shown, the TSS folder contains all TSS library files.

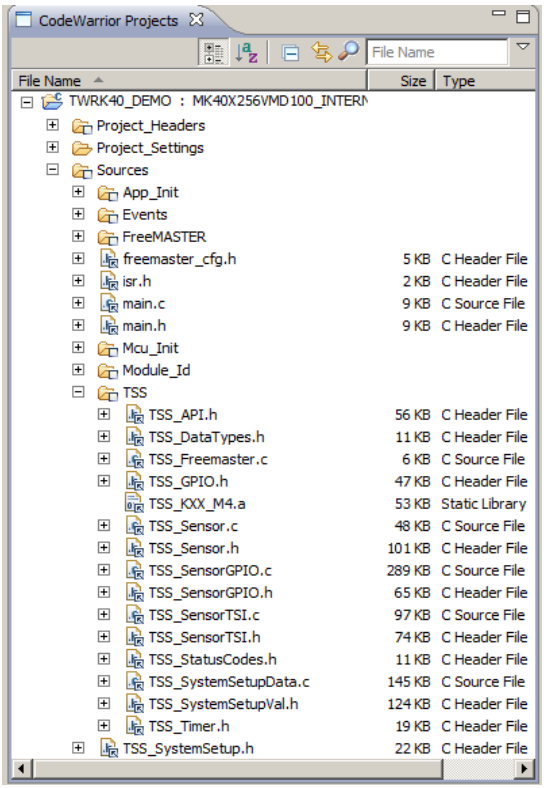

Figure 1-2. Touch sensing software project

1.4 System base modules

This section provides an overview of the libary modules and the source and header files that constitute these modules. The user configuration and API of each module are explained in Chapter 2, “TSS System setup parameters.”

1.4.1 System setup module

Module description

The system setup module enables a compile time configuration interface. In this module, a user can configure a set of parameters that require a pre-compile definition, such as, electrode numbers, MCU pins for each electrode, measurement method used by each electrode, and number of controls.

Module files

This module contains the following files in open source:

- TSS_SystemSetup.h — Edit the TSS_SystemSetup.h file to customize it for the application particular electrode structure requirements.
- TSS_SystemSetupVal.h — This file checks whether the application configuration, defined in the TSS_SystemSetup.h file, is consistent. Do not edit the TSS_SystemSetupVal.h file.
- TSS_SystemSetupData.c — This file creates the variables required for the TSS library and dependent on the electrode structure of a particular application. Do not edit this file.

1.4.2 GPIO module

Module description

The GPIO module provides an interface between the upper layers of the library and the MCU GPIO pins. This GPIO module is used by low-level routines for the default GPIO measurement method (detecting timeout condition). It provides the following functionalities:

- Configure GPIO as input
- Configure GPIO as output
- Read GPIO value
- Set GPIO bit value (write a 1 to the GPIO bit on its specific data register)
- Clear GPIO bit value (write a 0 to the GPIO bit on its specific data register)
- Determine GPIO electrode port register address and mask of this pin
- Set GPIO pin output strength high or low
- Read GPIO pin output strength value
- Set GPIO pin slew rate high or low
- Read GPIO pin slew rate

Module files

The TSS_GPIO.h file implements the GPIO module. It defines macros to configure and access the MCU I/O pin values, pin output strength, and slew rate settings. The upper layer uses these macros to manipulate the MCU I/O pins.

1.4.3 Hardware timer module

Module description

The hardware timer module provides the interface between the upper layers and the hardware timer specified at compile time. This timer module is used by low-level routines for the default GPIO method to detect a timeout condition.

This module covers the following functions:

- Timer configuration
  - Enable and disable interrupt
  - Set prescaler
  - Set and read timer module
- Timer start
- Timer stop
- Timer reset
- Set timer limit
- Read timer count
- Read 8 bits (low and high) of a timer count

- Read 16 bits of a timer count
- Hardware timer interrupt vector number assignment
- TPM, FTM, and MTIM8 timer modules are supported

Module files

The TSS_Timer.h file implements macros in the source code, which are required to access the timer hardware. The hardware timer interrupt service routine is implemented in the TSS_Sensor.c file.

1.5 Signal sensing modules

This section provides an overview of the low-level capacitance measurement modules and the source and header files that constitute these modules.

1.5.1 GPIO low level sensor method

Method description

The GPIO low level sensing method measures the capacitance by using the GPIO capacitive touch sensing method described in the *Touch Sensing User Guide*. This method is located in source code and provides the following functions:

- Configuration of all electrodes to a default state. The default state is output-high to achieve lower power consumption with external pull-up resistors.
- Initialization of the GPIO sensor method
- Capacitance measurement of a specific electrode using GPIO method
- Hardware timer interrupt handling
- Driving an electrode pin as low-output state if a timer charge timeout is detected (probably electrode shorted to the ground)
- Noise amplitude filter function

To improve noise immunity and increase system sensitivity, the electrode sampling function can integrate several subsequent electrode sampling values.

The HCS08 and ColdFire V1 version of the TSS library automatically allocate the TSS_HWTimerIsr(void) interrupt service routine for the appropriate interrupt vector. The ColdFire+, ARM Cortex-M0, and ARM Cortex-M4 version of the TSS library require manual allocation of the interrupt service routine.

Method files

The TSS_SensorGPIO.c, TSS_SensorGPIO_def.h, and TSS_SensorGPIO.h files implement the GPIO low level sensor method. This method uses the GPIO and the timer modules.

1.5.2 TSI low level sensor method

Method description

The TSI low level sensor method uses the TSI hardware (HW) peripheral module. Each TSI pin implements the capacitive measurement of an electrode having individual programmable detection thresholds and result registers. The TSI module can be functional in several low-power modes and it can be used to wake the CPU when a touch-event or a proximity-event is detected. The TSI method provides an interface between the upper layers and the hardware. This module has the following functions:

- TSI HW module initialization
- TSI HW module autocalibration
- MCU wakes from low power mode by a touch detection
- Triggering in all available modes (ALWAYS, software (SW), and AUTO)
- Obtaining measured values

The HCS08 and ColdFire V1 version of the TSS library automatically allocate TSS_TSIxIsr(void) interrupt service routine in the appropriate interrupt vector. The ColdFire+, ARM Cortex-M0+, and ARM Cortex-M4 version of the TSS library requires manual allocation of the interrupt service routine.

Method files

The files TSS_SensorTSI.c, TSS_SensorTSI.h and TSS_SensorTSI_def.h implement the TSI sensing method.

1.5.3 TSI Lite low level sensor method

Method Description

The TSI Lite Low-Level Sensor method is a simplified algorithm for the second generation of the Touch Sensing Input (TSI) HW peripheral module. Unlike the “full” TSI method and ARM Cortex-M0+ version of TSI Lite, the HCS08 version does not support waking the device from the low power mode. The TSI method provides an interface between the upper layers and the hardware. This module has the following functions:

- TSI HW module initialization
- TSI HW module autocalibration
- MCU wakes from low power mode by a touch detection on ARM Cortex-M0 version of TSI Lite
- Triggering in all available modes (ALWAYS, SW, and AUTO with external RTC, or LPTMR trigger source)
- Starting the capacitance measurement on the pin
- Starting the noise measurement on TSI modules supporting noise mode (Kinetis L ARM Cortex-M0+ family)
- Obtaining measured values

The HCS08 version of the TSS library automatically allocates TSS_TSIxIsr(void) interrupt service routine in the appropriate interrupt vector. The ARM Cortex-M0 version of the TSS library requires manual allocation of the interrupt service routine.

Method Files

The files TSS_SensorTSIL.c, TSS_SensorTSIL_def.h and TSS_SensorTSIL.h implement the TSI Lite method.

1.6 Key detector modules

The key detector modules determine which electrodes are pressed or released based on the values obtained by the capacitive sensing layer.

The key detector modules also detect, report, and act on fault conditions during the scanning process. The two main fault conditions are electrode short-circuit to supplying voltage or grounding. The same conditions can be caused by a small capacitance (equal to a short circuit to supply voltage) or by a large capacitance (equals to a short circuit to ground).

The main output of these modules is the indication of a finger presence or an absence in each of the active electrodes.

The current version of TSS provides these key detector methods:

- Basic key detector
- AFID key detector
- Noise key detector

1.6.1 Basic key detector module

Module description

Module description implements algorithms for identification of touched and released electrodes by obtaining a baseline value (which is the "DC" value of the capacitance when a finger is not present) and comparing it to the immediate capacitance value. A user can manually select this key detector in the TSS configuration.

The baseline is obtained by a low pass IIR-based filter which removes slow environment changes that could cause a false electrode press detection, such as, air humidity, temperature, and other external variables. A de-bounce function is also implemented in this module to eliminate false detections caused by instantaneous noise.

The shielding functionality is also processed in the module. The shielding feature uses a shielding electrode, which measures the overall environment noise, to compensate for a signal drift on an electrode. It eliminates low frequency noise modulated on the capacitance signal and protects the guarded system from detecting false touches caused by water drops.

The key detector provides an automatic sensitivity calibration function. The function periodically adjusts the level of electrode sensitivity, which is calculated according to the estimated noise level and touch-tracking information. Although the sensitivity no longer has to be set manually, the settings are still available for more precise tuning.

Main functions of the basic key detector module:

- Active electrodes detection pressed or not pressed
- Electrode detection debouncing

- Baseline generation
- IIR filtering of current capacitance signal
- Shielding
- Proximity detection
- Sensitivity Autocalibration
- Electrode status reporting
- Fault reporting

Module files

The key detector module is implemented in the object code integrated inside the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, TSS_KXX_M4_FPU.a, and TSS_KXX_M0.a/.lib library files.

For more information regarding the electrode touch detection method, see the *Touch Sensing User Guide*.

1.6.2 AFID key detector module

Module description

The AFID (Advanced Filtering and Integrating Detection) key detector operates by using two IIR filters with different depths (one short/fast, the other long/slow) and, then, integrating the difference between the two filtered signals. Although this algorithm is more immune to noise, it is not compatible with other noise-cancellation techniques such as shielding. The AFID key detector can be manually selected in the TSS configuration.

The key detector also provides an automatic-sensitivity calibration. The calibration periodically adjusts the level of electrode sensitivity, which is calculated according to touch-tracking information. Although the sensitivity no longer has to be manually set, the settings are still available for more precise tuning.

The standard baseline, which is set according to the low pass IIR filter, is also calculated for analog decoders and proximity function. A de-bounce function is implemented in this module to eliminate false detections caused by instantaneous noise.

Main functions of the AFID key detector module:

- Two filters with integration for touch detection
- Electrode detection debouncing
- Baseline generation
- IIR filtering of current capacitance signal
- Proximity detection
- Sensitivity autocalibration
- Electrode status reporting
- Fault reporting

Module files

The key detector module is implemented in the object code integrated inside the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, TSS_KXX_M4_FPU.a, and TSS_KXX_M0.a/.lib library files.

For more information regarding the electrode touch detection method, see the *Touch Sensing User Guide*.

1.6.3 Noise key detector module

Module description

This key detector processes noise signals only from the TSI module which supports the noise mode. This feature is currently supported by TSI modules in Kinetis L ARM Cortex-M0+ family. The noise mode feature is an alternative hardware measurement method intended for touch detection in an extremely noisy environment.

The noise key detector cannot be selected as a standalone key detector. Instead, it is intended to be a supplement for either Basic or AFID key detectors. The measurement and the touch detection are executed automatically at a defined rate. The key detector provides only the initial sensitivity calibration function. The sensitivity is not updated after the initial calibration.

The algorithm provides the touch information only and is not compatible with analog decoders, low power, shielding, or proximity features. A de-bounce function is implemented in this module to eliminate false detections caused by instantaneous noise.

Main functions of the noise key detector module:

- Switching between capacitance mode and noise mode, and measurement scheduling
- Electrode detection debouncing
- IIR filtering of a current noise signal
- Initial sensitivity calibration
- Fault reporting

Module files

The key detector module is implemented in the object code integrated inside the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, TSS_KXX_M4_FPU.a, and TSS_KXX_M0.a/.lib library files.

For more information regarding the electrode touch detection method, see the *Touch Sensing User Guide*.

1.7 Decoder module

Module Description

Decoders provide the highest level of abstraction in the library. In this layer, the information regarding touched and untouched electrodes is interpreted and shows the status of a control in a behavioral way. Additional functionalities can be provided by the decoders, such as FIFOs. Note that the decoder-related code exists only once in memory, which implies that, despite the number of rotary controls in the system, only one rotary decoder resides in the memory. Decoders can be described as classes of an object-oriented language where each control has a decoder associated with it. Therefore, the control becomes an instance of the decoder (an object). However, not all decoders are necessarily instantiated in every system.

Decoder types supported by the library:

- Rotary

- Slider
- Keypad
- Analog rotary
- Analog slider
- Matrix

Module Files

The decoder module is implemented in the object code integrated in the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, TSS_KXX_M4_FPU.a, and TSS_KXX_M0.a/.lib library files. For more information about the Decoders functionality, see the *Touch Sensing User Guide*.

1.8 System configuration and management module

Module description

This module implements the configuration and management functions required to integrate the library in the user application. The module contains the following functions:

- `TSS_Init()`— This function initializes the TSS library.
- `TSS_Task()`— The function must be called periodically by the user application to provide the CPU time to the TSS library. All electrodes are processed during a single execution of this function, but the measured data are evaluated after at least two executions. The process status is reported by the return value.
- `TSS_TaskSeq()`— This function is an alternative to the `TSS_Task()` function. It must be called periodically by the user application to provide the CPU time to the TSS library. One electrode is processed in one function execution. When the function is executed periodically, it processes all electrodes and decoders. The process status is reported by the return value.
- `TSS_SetSystemConfig()`— Provides configuring the TSS library during run-time.
- `TSS_GetSystemConfig()`— Provides reading of the TSS library registers during run-time.

If this module is enabled, it can perform integrity verification of the TSS RAM variables by executing a Checksum check. The verification is embedded in all functions provided by this module.

Module files

The system configuration and management module is implemented in the object code integrated in the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, TSS_KXX_M4_FPU.a, and TSS_KXX_M0.a/.lib library files.

The TSS_API.h file provides headers and macros required to integrate functions and variables defined in the library files.

Chapter 2
TSS System setup parameters

Table 2-1 provides a summary of the TSS static configuration options available in the TSS_SystemSetup.h file. For more information regarding the each configuration parameter, see the *Touch Sensing User Guide*.

Table 2-1. System setup parameters

Parameter	Range	Description	Used
Features Configuration			
TSS_USE_KEYDETECTOR_VERSION	1 Basic 2 AFID	Defines main key detector method	Optional, default 1
TSS_USE_NOISE_MODE	0–1	Enables the noise key detector as supplement of main keydetector defined by key detector version parameter	Optional, default 0. Available only if TSI module supports noise mode
TSS_USE_SIMPLE_LOW_LEVEL	0–1	Enables the Simple Low Level routines option	Optional, default value depends on availability for the MCU
TSS_USE_DELTA_LOG	0–1	Enables the instant delta feature	Optional, default 0
TSS_USE_SIGNAL_LOG	0–1	Enables the instant signal feature	Optional, default 0
TSS_USE_INTEGRATION_DELTA_LOG	0-1	Enables the instant integration signal from AFID key detector	Optional, default 0
TSS_USE_NOISE_SIGNAL_LOG	0–1	Enables the instant noise signal	Optional, default 0
TSS_USE_FREEMASTER_GUI	0–1	Enables FreeMASTER GUI support	Optional, default 0
TSS_USE_CONTROL_PRIVATE_DATA	0–1	Enables Control Private Data feature	Optional, default 0
TSS_USE_AUTO_SENS_CALIBRATION	0–1	Enables automatic sensitivity calibration	Optional, default 0
TSS_USE_AUTO_SENS_CALIB_INIT_DURATION	0–255	Sets duration of an automatic sensitivity calibration initialization	Optional, default 100
TSS_USE_AUTO_SENS_CALIB_LOW_LIMIT	0–1	Enables automatic sensitivity low limit	Optional, default 0
TSS_USE_BASELINE_INIT_DURATION	0–255	Sets duration of a baseline initialization	Optional, default 0
TSS_USE_LOWPOWER_THRESHOLD_BASELINE	0-1	Enables to use dc-tracker baseline for calculation of low power wake-up threshold	Optional, default 0

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_USE_LOWPOWER_CALIBRATION	0-1	Enables workaround for low power errata on TSI modules in Kinetis silicon versions 2 and 3	Optional, default 0
TSS_USE_AFID_FAST_FILTER_RATIO	4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 240, 260, 280, 300, 350, 400, 500, 600, 700, 800, 900, 1000	Defines weight of the fast filter in AFID keydetector in the form of ratio f_{sample}/f_{cutoff}	Optional, default 50
TSS_USE_AFID_SLOW_FILTER_RATIO	AFID Fast Filter Ratio - 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 240, 260, 280, 300, 350, 400, 500, 600, 700, 800, 900, 1000	Defines weight of the slow filter in AFID keydetector in the form of ratio f_{sample}/f_{cutoff}	Optional, default 200
Signal Control Options			
TSS_USE_GPIO_STRENGTH	0–1	Enables strength on usable pins for GPIO method.	Optional, default 0. Used only for the GPIO method.
TSS_USE_GPIO_SLEW_RATE	0–1	Enables slew rate on usable pins for GPIO method.	Optional, default 0. Used only for the GPIO method.
TSS_USE_IIR_FILTER	0–1	Enables IIR filter	Optional, default 0
TSS_USE_NOISE_AMPLITUDE_FILTER	0–1	Enables noise amplitude filter for GPIO method.	Optional, default 0. Used only for the GPIO method.
TSS_USE_SIGNAL_SHIELDING	0–1	Enables Signal Shielding function	Optional, default 0
TSS_USE_SIGNAL_DIVIDER	0–1	Enables signal divider	Optional, default 0
TSS_USE_SIGNAL_MULTIPLIER	0–1	Enables signal multiplier	Optional, default 0
TSS_USE_DEFAULT_ELECTRODE_LEVEL_LOW	0–1	Sets low electrode level between measurements for GPIO method.	Optional, default 0. Used only for the GPIO method.
Callbacks Definition			

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_ONFAULT_CALLBACK	Any valid function name. This callback function must be provided by the user.	This is the name of a function that matches the OnFault callback prototype	Optional. If the macro is not defined, the callback is disabled.
TSS_ONINIT_CALLBACK	Any valid function name. This callback function must be provided by the user.	This is the name of a function that matches the OnInit callback prototype	Optional. If macro is not defined, 'TSS_fOnInit' name is used.
TSS_ONPROXIMITY_CALLBACK	Any valid function name. This callback function must be provided by the user.	This is the name of a function that matches the OnProximity callback prototype	Optional. If macro is not defined, the proximity function is disabled.
Definition of Function Control Source			
TSS_USE_AUTOTRIGGER_SOURCE	RTC, LPTMR, TSI, TSI0, TSI1, UNUSED	Name of the hardware device used for management of TSS AUTO triggering	Optional, default UNUSED
TSS_USE_LOWPOWER_CONTROL_SOURCE	TSI, TSI0, TSI1, UNUSED	Name of hardware device used for management of the Low Power mode	Optional, default UNUSED
Code Size Reduction Options			
TSS_USE_DATA_CORRUPTION_CHECK	0–1	Enables compilation of TSS RAM data consistency check function	Optional, default 1
TSS_USE_TRIGGER_FUNCTION	0-1	Enables compilation of Triggering function	Optional, default 0
TSS_USE_STUCK_KEY	0-1	Enables compilation of Stuck Key detection function	Optional, default 1
TSS_USE_NEGATIVE_BASELINE_DROP	0-1	Enables compilation of Negative Baseline Drop function	Optional, default 1
TSS_USE_AUTO_HW_RECALIBRATION	0-1	Enables automatic hardware recalibration	Optional, default 1
Debug Options			
TSS_ENABLE_DIAGNOSTIC_MESSAGES	0–1	Enables diagnostic messages during compilation	Optional, default 0
Electrodes Configuration			
TSS_N_ELECTRODES	1–64	Sets the number of electrodes to be used by the application	Always

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_Ex_P	Any valid GPIO port on the MCU	Configures the MCU GPIO port to be used for each electrode	Always, except TSI module electrode
TSS_Ex_B	Any valid GPIO pin on the MCU	Configures the MCU GPIO pin to be used for each electrode	Always, except TSI module electrode
TSS_Ex_TYPE	GPIO, TSInCHm	Determines the measurement method for an electrode	Optional, default GPIO
TSS_Ex_NOISE_AMPLITUDE_FILTER_SIZE	2–255	Determines the size of the noise amplitude filter for an electrode	Optional, default 255
TSS_Ex_SHIELD_ELECTRODE	0-63	Determines the number of shielding electrode assigned to this electrode	Optional, if defined then the electrode is shielded.
TSS_Ex_SIGNAL_DIVIDER	0-255	Determines the value of signal divider for this electrode	Optional, if defined or non zero then the electrode uses divider.
TSS_Ex_SIGNAL_MULTIPLIER	0-255	Determines the value of signal multiplier for this electrode	Optional, if defined or non zero then the electrode uses multiplier.
Controls Configuration			
TSS_N_CONTROLS	0-16	Sets the number of controls to be used by the application	Always
TSS_Cn_TYPE	TSS_CT_KEYPAD TSS_CT_SLIDER TSS_CT_ROTARY TSS_CT_ASLIDER TSS_CT_AROTARY TSS_CT_MATRIX	Determines the type of the control	Always, if Controls > 0
TSS_Cn_INPUTS	Any valid array with number of items: 1-16 for keypad 2-16 for slider 3-16 for rotary 2-16 for analog slider 3-16 for analog rotary 3-16 for matrix	Determines the electrodes that composes the control	Always, if Controls > 0
TSS_Cn_INPUTS_NUM_X	2-14	Determines the amount of electrodes that compose the matrix control on X axis	Always, if matrix control is used

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_Cn_STRUCTURE	Any valid name selected by the user	Indicates the name of the configuration and status structure of the control. It is used to create the configuration and status structure in a different file. This structure is not declared by users.	Always, if Controls > 0
TSS_Cn_CALLBACK	Any valid function name. This callback function is created by the user.	This is the name of a valid function that matches the callback prototype	Always, if Controls > 0
TSS_Cn_KEYS	Any valid array defined by an user with maximum length 65536 items	Defines the groups of electrodes for group decoder. Valid only for keypad control.	Optional.
Peripheral Specific Configuration			
TSS_HW_TIMER	TPMx, FTMx, MTIMx	Name of hardware timer used for GPIO method.	Always for GPIO method.
TSS_SENSOR_PRESCALER	NA, depends on used timer	Prescaler for all used timers	Optional, default 2. If the GPIO method is used.
TSS_SENSOR_TIMEOUT	128-65535, except HCS08 where 128-511(2047 if IIR filter is disabled) is used	Defines timeout for all used timers	Optional, default 511. If the GPIO method is used.
TSS_TSI_RESOLUTION	1–16 bits	Defines resolution of TSI in bits for autocalibration	Optional, default 11. If the TSI module is used.
TSS_TSI_EXTCHRG_LOW_LIMIT	0 — TSI module EXTCHRG range	Defines low limit of EXTCHRG for TSI autocalibration	Optional, default 0. If the TSI module is used.
TSS_TSI_EXTCHRG_HIGH_LIMIT	EXTCHRG low limit — TSI module EXTCHRG range	Defines high limit of EXTCHRG for TSI autocalibration	Optional, default 7. If the TSI module is used.
TSS_TSI_PS_LOW_LIMIT	0–7	Defines low limit of PS for TSI autocalibration	Optional, default 0. If the TSI module is used.
TSS_TSI_PS_HIGH_LIMIT	PS low limit – 7	Defines high limit of PS for TSI autocalibration	Optional, default 7. If the TSI module is used.
TSS_TSI_AMCLKS	0 Bus Clock 1 MCGIRCLK 2 OSCERCLK 3 Not valid	Defines TSI Active mode Clock Source	Optional, default 0. If the TSI module supports AMCLKS function. If the TSI module is used.

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_TSI_AMCLKDIV	0 divider set to 1 1 divider set to 2048	Defines TSI Active Mode Clock Divider	Optional, default 1. if the TSI module supports AMCLKDIV register. If the TSI module is used.
TSS_TSI_AMPSC	0 divided by 1 1 divided by 2 2 divided by 4 3 divided by 8 4 divided by 16 5 divided by 32 6 divided by 64 7 divided by 128	Defines TSI Active Mode Prescaler	Optional, default 0. If TSI module supports AMPSC register. If the TSI module is used.
TSS_TSI_LPCLKS	0 LPOCLK 1 VLPOSCCLK	Defines TSI Low Power Mode Clock Source	Optional, default 0. If TSI module supports LPCLKS register. If the TSI module is used.
TSS_TSI_DVOLT	NA, depends on used TSI module	Defines TSI Delta Voltage value	Optional, with default maximum value. If TSI module provides DVOLT register.
TSS_TSI_SMOD	0-255	Defines TSI Scan Modulo	Optional, default 0. If TSI module provides SMOD register.
TSS_TSI_LPSCNITV	0 for 1 ms 1 for 5 ms 2 for 10 ms 3 for 15 ms 4 for 20 ms 5 for 30 ms 6 for 40 ms 7 for 50 ms 8 for 75 ms 9 for 100 ms 10 for 125 ms 11for 150 ms 12 for 200 ms 13 for 300 ms 14 for 400 ms 15 for 500 ms	Defines TSI Low Power Mode Scan Interval	Optional, default 0. If TSI module provides LPSCNITV register.
TSS_TSI_NOISE_PS	0 divided by 1 1 divided by 2 2 divided by 4 3 divided by 8 4 divided by 16 5 divided by 32 6 divided by 64 7 divided by 128	Defines External Oscillator Prescaler for TSI noise mode	Optional, default 3. If TSI module provides PS register and noise mode.

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_TSI_NOISE_FILTER	0 for 3 bits 1 for 2 bits 2 for 1 bit 3 for disabled	Defines Number Of Filter Bits for TSI noise mode	Optional, default 0. If TSI module provides EXTCHRG register and noise mode.
TSS_TSI_NOISE_RS	0 for 32 kOhm 1 for 187 kOhm	Defines Series Resistance for TSI noise mode	Optional, default 0. If TSI module provides EXTCHRG register and noise mode.
TSS_TSI_NOISE_REFCHRG	0 for 500 nA 1 for 1 ?A 2 for 2 ?A 3 for 4 ?A 4 for 8 ?A 5 for 16 ?A 6 for 32 ?A 7 for 64 ?A	Defines Reference Oscillator Charge Current for TSI noise mode	Optional, default 0. If TSI module provides REFCHRG register and noise mode.
Proximity Specific Configuration			
TSS_SENSOR_PROX_PRESCALER	NA, depends on used timer	Prescaler for all used timers in proximity mode	Optional, default 2. If the GPIO method is used.
TSS_SENSOR_PROX_TIMEOUT	128-65535, except HCS08 where 128-511(2047 if IIR filter is disabled) is used	Defines timeout for all used timers in proximity mode	Optional, default 511. If the GPIO method is used.
TSS_TSI_PROX_RESOLUTION	1–16 bits	Defines resolution of TSI in bits for autocalibration in proximity mode	Optional, default 11. If the TSI module is used.
TSS_TSI_PROX_EXTCHRG_LOW_LIMIT	0 — TSI module EXTCHRG range	Defines low limit of EXTCHRG for TSI autocalibration in proximity mode	Optional, default 0. If the TSI module is used.
TSS_TSI_PROX_EXTCHRG_HIGH_LIMIT	0 — TSI module EXTCHRG range	Defines high limit of EXTCHRG for TSI autocalibration in proximity mode	Optional, default 7. If theTSI module is used.
TSS_TSI_PROX_PS_LOW_LIMIT	0–7	Defines low limit of PS for TSI autocalibration in proximity mode	Optional, default 0. If the TSI module is used.
TSS_TSI_PROX_PS_HIGH_LIMIT	0– 7	Defines high limit of PS for TSI autocalibration in proximity mode	Optional, default 7. If the TSI module is used.

Chapter 3
Application Interface

This section describes the TSS library initialization task function and its usage. It also describes the configuration and status structure of the library and how to manipulate it by using the **TSS_SetSystemConfig** function.

Each control in the application has a configuration and a status structure assigned to it. Control parameters depend on the control type. This section describes the following:

- Configuration and status structure for each control type
- How to manipulate configuration and status structures by using the control configuration function
- Callback functions for each control type

The TSS_API.h file contains the function prototypes and variables declaration of the application interface. This file must be included in the applications source code that manages the TSS library.

3.1 TSS initialization function

This function initializes the data structures and low level routines of the library with default values. The OnInit callback is executed during the function call. The MCU and peripherals clock must be configured before calling the `TSS_Init()`function or before the OnInit callback.

Function prototype

The TSS initialization function has the following prototype defined in the TSS_API.h file:

```
uint8_t TSS_Init(void);
```

Input parameters

None

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	TSS initialization has successfully finished
TSS_STATUS_RECALIBRATION_FAULT	The fault occurred during the recalibration of low level hardware

3.2 TSS task function

The application calls this function periodically to provide the CPU time for the TSS library. The `TSS_Task()` routine handles measurement initialization by acquiring the sample and processing the capacitance signal values. Depending on the measurement method selected for given electrodes, the physical sampling is either performed by the Hardware module (for example TSI) or directly by the `TSS_Task()` routine. This influences the duration of the `TSS_Task()` execution and the number of times

it must be called before one full set of samples is taken and processed. Processing status is reported by the `TSS_Task()` return value.

Function prototype

The TSS task function has the following prototype defined in the TSS_API.h file:

```
uint8_t TSS_Task(void);
```

Input parameters

None

Return value

The return value is an unsigned byte with the following possible return values, as defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	TSS task finished the measurement of all electrodes and has evaluated the values
TSS_STATUS_PROCESSING	Measurement of electrodes in progress or the TSS task has not finished the evaluation
TSS_STATUS_BUSY	TSS_Task() was executed during the TSS register changes performed by TSS_SetSystemConfig().

NOTE

It is not recommended to perform any TSS register changes during electrode processing until the `TSS_Task()` returns TSS_STATUS_OK.

3.3 TSS task sequenced function

The function `TSS_TaskSeq()` is a more “atomic” alternative to `TSS_Task()`. The application calls the `TSS_TaskSeq()`periodically to provide the CPU time to the TSS library. Only one electrode is processed during a single call. Therefore, the execution takes less time and enables a smoother multitasking for other processes which are running simultaneously with the `TSS_Task()`. The two-phase processing principle for electrodes is the same as processing in the `TSS_Task()`. Because the function is periodically executed, it processes all electrodes and decoders.

Function prototype

The TSS task sequenced function has the following prototype defined in the TSS_API.h file:

```
uint8_t TSS_TaskSeq(void);
```

Input parameters

None

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	TSS sequenced task finished measurement of all electrodes and evaluated the values
TSS_STATUS_PROCESSING	Measurement of electrodes is in progress or the TSS sequenced task has not finished the evaluation
TSS_STATUS_BUSY	TSS_TaskSeq() was executed during the TSS register changes performed by TSS_SetSystemConfig().

NOTE

It is not recommended to perform any TSS register changes during electrode processing until the `TSS_TaskSeq()` returns TSS_STATUS_OK.

3.4 TSS Library Configuration Save and Restore Functions

The TSS library provides functions for saving and restoring the TSS configuration. The purpose of these functions is to recover the TSS configuration after low power wake up which causes an MCU to reset. There are several functions available for saving and restoring the TSS configuration. The following chapters describe each function.

3.4.1 Save and restore electrode data

The electrode data consists of the electrode sensitivity and the baseline values in the basic keydetector. The data consist of immediate filter values, electrode sensitivity, and baseline values in the AFID keydetector.

To save the data of a single electrode, the TSS_StoreElectrodeData function must be called at any time. The function is declared in the TSS_API.h file.

Function prototype

```
int16_t TSS_StoreElectrodeData(uint8_t u8Electrode, void *pDestAddr, uint16_t u16Length)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	u8Electrode	Any of the electrode number within maximum used number of electrodes.	Parameter defines electrode number which configuration will be saved
void*	pDestAddr	Valid address of the destination memory area	Pointer to begin of memory area where data will be saved
uint16_t	u16Length	Valid number with unsigned integer range	Size of available memory area in bytes

Return value

The return value is either a signed integer value defining number of saved bytes, or return error status values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned number	If return value is higher than u16Length parameter, more memory is needed and the operation is unsuccessful. Otherwise, the return value defines a number of saved bytes and the operation is successful.
TSS_ERROR_DATASYS_OUT_OF_RANGE	An electrode number is higher than the maximum number of electrodes.
TSS_ERROR_DATASYS_MEM_NULL	Pointer address is not valid.

To restore the data of a single electrode, the TSS_LoadElectrodeData function must be called. The function can be called only during TSS initialization time, between TSS_Init call and the first TSS_Task call. The function is declared in the TSS_API.h file.

Function prototype

```
int16_t TSS_LoadElectrodeData(uint8_t u8Electrode, void *pSourceAddr)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	u8Electrode	Any electrode number within maximum used number of electrodes	Parameter defines an electrode number which restores the configuration.
void*	pSourceAddr	Valid address of the source memory area	A pointer to the beginning of the memory area where the source data is stored.

Return value

The return value is either a signed integer value defining a number of saved bytes, or return error status values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned number	Return value defines a number of saved bytes and the operation is successful.
TSS_ERROR_DATASYS_OUT_OF_RANGE	Electrode number is higher than the maximum number of electrodes.
TSS_ERROR_DATASYS_MEM_NULL	Pointer address is not valid.
TSS_ERROR_DATASYS_READ_ONLY	TSS is not in the initialization phase between TSS_Init call and the first TSS_Task call.

3.4.2 Save and restore module data

Module data is a configuration of all used low-level measurement modules. If a TSI module is used, the data includes a configuration of the TSI autocalibration. The GPIO method does not provide this data.

To save the used module data, the TSS_StoreModulesData function must be called. The function can be called at any time and is declared in the TSS_API.h file.

Function prototype

```
int16_t TSS_StoreModulesData(void *pDestAddr, uint16_t u16Length)
```

Input parameters

Type	Name	Valid Range and Values	Description
void*	pDestAddr	Valid address of the destination memory area	A pointer to the beginning of the memory area where data is saved.
uint16_t	u16Length	Valid number with unsigned integer range	A size of the available memory area in bytes.

Return value

The return value is either a signed integer value defining a number of saved bytes, or a possible return error status value defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned number	If a return value is higher than the u16Length parameter, more memory is needed and the operation is unsuccessful. Otherwise, a return value defines a number of saved bytes and operation is successful.
TSS_ERROR_DATASYS_MEM_NULL	A pointer address is not valid.

To restore the module data, the TSS_LoadModulesData function must be called. The function can be called only during TSS initialization time, between TSS_Init call and the first TSS_Task call and is declared in the TSS_API.h file.

Function prototype

```
int16_t TSS_LoadModulesData(void *pSourceAddr)
```

Input parameters

Type	Name	Valid Range and Values	Description
void*	pSourceAddr	Valid address of the source memory area	Pointer to begin of memory area where are source data stored

Return value

The return value is either a signed integer value defining a number of saved bytes, or return error status values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned number	A return value defines a number of saved bytes and the operation is successful.
TSS_ERROR_DATASYS_MEM_NULL	A pointer address is not valid.
TSS_ERROR_DATASYS_READ_ONLY	TSS is not in the initialization phase between TSS_Init call and the first TSS_Task call.

3.4.3 Save and restore all system data

The system data consist of electrode and module data. The data is described in Section 3.4.1, “Save and restore electrode data” and Section 3.4.2, “Save and restore module data”.

To save the system data, the TSS_StoreAllSystemData function must be called. The function can be called at any time and is declared in the TSS_API.h file.

Function prototype

```
int16_t TSS_StoreAllSystemData(void *pDestAddr, uint16_t u16Length)
```

Input parameters

Type	Name	Valid Range and Values	Description
void*	pDestAddr	Valid address of the destination memory area	A pointer to the beginning of the memory area where data is saved.
uint16_t	u16Length	Valid number with unsigned integer range	The size of available memory area in bytes.

Return value

The return value is either a signed integer value defining number of saved bytes, or return error status values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned number	If the return value is higher than the u16Length parameter, more memory is needed and operation is unsuccessful. Otherwise, the return value defines a number of saved bytes and operation is successful.
TSS_ERROR_DATASYS_MEM_NULL	A pointer address is not valid.

To restore the system data, the TSS_LoadAllSystemData function must be called. The function can be called only during TSS initialization time, between TSS_Init call and the first TSS_Task call and is declared in the TSS_API.h file.

Function prototype

```
int16_t TSS_LoadAllSystemData(void *pSourceAddr)
```

Input parameters

Type	Name	Valid Range and Values	Description
void*	pSourceAddr	Valid address of the source memory area	A pointer to the beginning of the memory area where source data is stored.

Return value

The return value is either a signed integer value defining number of saved bytes, or return error status values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned number	The return value defines a number of saved bytes and the operation is successful.
TSS_ERROR_DATASYS_CHECKSUM	Data in the source memory area is corrupted.
TSS_ERROR_DATASYS_MEM_NULL	A pointer address is not valid.
TSS_ERROR_DATASYS_NOT_ENOUGH_DATA	Not enough data is provided in the source memory area.
TSS_ERROR_DATASYS_READ_ONLY	The TSS is not in an initialization phase between TSS_Init call and the first TSS_Task call.

3.5 TSS Library Configuration and Status Registers

The TSS library Control and Status (C&S) registers constitute the main application interface to the TSS library. The application may use the registers to customize the library operation and determine its status.

Although the C&S registers are located in RAM, they are read-only and declared as constant values. This prevents a user from modifying status values or corrupting them with invalid configuration values.

3.5.1 Writing to the Configuration and Status Registers

The TSS_SetSystemConfig function is called to change values in the Configuration and Status registers. If necessary, the function can reinitialize the low-level hardware. The function is declared in the TSS_API.h file.

Function prototype

```
uint8_t TSS_SetSystemConfig(uint8_t u8Parameter, uint16_t u16Value)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	u8Parameter	Any of the parameter codes provided by the configuration and status management	Code indicating the parameter configured
uint16_t	u16Value	Depends on the specific parameter configured	The new desired value for the respective configuration register

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	Configuration was done successfully
TSS_ERROR_CONFSYS_BUSY	TSS_SetSystemConfig() was executed during the electrode measurement.

Return Value	Description
TSS_ERROR_CONFSYS_ILLEGAL_PARAMETER	The configuration is not successful due to illegal parameter number.
TSS_ERROR_CONFSYS_READ_ONLY_PARAMETER	The configuration is not successful because the user attempted to modify a read-only parameter.
TSS_ERROR_CONFSYS_OUT_OF_RANGE	The configuration is not successful because the new value is out of range.
TSS_ERROR_CONFSYS_LOWPOWER_SOURCE_NA	The configuration is not successful because the low-power control source device is not selected.
TSS_ERROR_CONFSYS_INCORRECT_LOWPOWER_EL	The configuration is not successful because the selected electrode does not belong to the low-power control source device.
TSS_ERROR_CONFSYS_TRIGGER_SOURCE_NA	The configuration is not successful because the trigger source device is not selected.
TSS_ERROR_CONFSYS_PROXIMITY_CALLBACK_NA	The configuration is not successful because the proximity callback is not defined.
TSS_ERROR_CONFSYS_RECALIBRATION_FAULT	The configuration is not successful because the fault occurred during the recalibration of the low level hardware.
TSS_ERROR_CONFSYS_SHIELD_NA	The configuration is not successful because the shielding functionality is not enabled.
TSS_ERROR_CONFSYS_HWRECALIB_PENDING	The configuration is not successful because the hardware recalibration is in progress.
TSS_ERROR_CONFSYS_MANRECALIB_PENDING	The configuration is not successful because the manual recalibration is in progress.
TSS_ERROR_CONFSYS_DCTRACKER_RATE_ZERO	The configuration is not successful because dc tracker rate register is set to zero value.
TSS_ERROR_CONFSYS_STUCKKEY_TIMEOUT_ZERO	The configuration is not successful because the stuckey timeout register is set to zero value.

3.5.2 Reading the Configuration and Status registers

To read values from the Configuration and Status register, the application can either access the system structures directly or it can use the TSS_GetSystemConfig function. The function is declared in the TSS_API.h file.

Function prototype

```
uint16_t TSS_GetSystemConfig(uint8_t u8Parameter)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	u8Parameter	Any of the parameter codes provided by the configuration and status management	Code indicating the parameter configured

Return value

The return value corresponds to an unsigned integer value of a register specified by the u8Parameter, or error values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned integer value	Value corresponds to an actual unsigned integer value of a register specified by u8Parameter
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Read was not successful due to illegal parameter
TSS_ERROR_GETCONF_ILLEGAL_CONTROL_TYPE	Read was not successful due to illegal control type

In the TSS register structure, Configuration and Status registers of the library are divided into two types. One set of registers is contained inside a structure, while the others are declared in global arrays. Register structure is defined as follows.

```
typedef struct{
        volatile const TSS_SYSTEM_FAULTS Faults;                        //Note 1
        volatile const TSS_SYSTEM_SYSCONF SystemConfig;                 //Note 2
        volatile const uint8_t NSamples;
        volatile const uint8_t DCTrackerRate;
        volatile const uint8_t SlowDCTrackerFactor;
        volatile const uint8_t ResponseTime;
        volatile const uint8_t StuckKeyTimeout;
        volatile const uint8_t LowPowerElectrode;
        volatile const uint8_t LowPowerElectrodeSensitivity;
        volatile const TSS_SYSTEM_TRIGGER SystemTrigger;                //Note 3
} TSS_CSSystem;
```

NOTE

1. The TSS_SYSTEM_FAULTS type is an eight bit-field structure described in Section 3.5.4, “Faults register"”
2. The TSS_SYSTEM_SYSCONF type is an 16 bit-field structure described in Section 3.5.5, “System Configuration register"”
3. The TSS_SYSTEM_TRIGGER type is an eight bit-field structure described in Section 3.5.13, “System Trigger register"”

The TSS library configuration and control structure instance is named as follows:

tss_CSSys

To read any variable inside this structure, use the standard C structure read statements. Access example of the Fault register in C:

```
Temp = tss_CSSys.Faults;
```

To access to the electrode-specific most critical configuration registers quickly, the library defines the following registers as global arrays.

- `const uint8_t tss_au8Sensitivity[TSS_N_ELECTRODES];`
- `const uint8_t tss_au8ElectrodeEnablers[((TSS_N_ELECTRODES–1)/8)+1];`
- `const uint8_t tss_au8ElectrodeStatus[((TSS_N_ELECTRODES–1)/8)+1];`
- `const uint8_t tss_au8ElectrodeDCTrackerEnablers[((TSS_N_ELECTRODES-1)/8)+1];`

To read any of these registers, use the standard C array read statements. Access an example of the au8Sensitivity register in C:

```
Temp = tss_au8Sensitivity[ Electrode ];
```

Although the registers can be read directly as the part of the configuration structure or from global arrays, they must be written to by using the TSS_SetSystemConfig() function. For more details, see Section 3.5.1, “Writing to the Configuration and Status Registers".

3.5.3 Configuration and Status registers list

Table 3-1. Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	Faults	Section 3.5.4, “Faults register"	0x00	RW — Shows the faults generated by the system.
0x01	2	SystemConfig	Section 3.5.5, “System Configuration register"	0x00	RW — Main configuration of the TSS library.
0x02	1	NSamples	Section 3.5.6, “Number of Samples register"	0x08	RW — Determines the number of samples scanned for a single measurement.
0x03	1	DCTrackerRate	Section 3.5.7, “DC Tracker Rate register"	0x64	RW — Determines how often the baseline is updated for common electrodes. This number represents the number of calls to the TSS task function required before dc tracker execution.
0x04	1	SlowDCTrackerFactor	Section 3.5.8, “Slow DC Tracker Factor register”	0x64	RW — Determines how often the baseline will be updated for proximity and shielding electrodes. This number is a multiplication factor used with standard DCTrackerRate register value .
0x05	1	ResponseTime	Section 3.5.9, “Response Time register"	0x04	RW — Configures the number of TSS task calls necessary to determine whether a key is pressed or not.
0x06	1	StuckKeyTimeout	Section 3.5.10, “Stuck-key Timeout register"	0x00	RW — Configures the number of TSS task calls necessary to detect a stuck key.
0x07	1	LowPowerElectrode	Section 3.5.11, “Low Power Electrode register"	0x00	RW — Number of electrodes scanned in low-power mode and proximity mode.
0x08	1	LowPowerElectrodeSensitivity	Section 3.5.12, “Low-Power Electrode Sensitivity register"	0x3F	RW — Sensitivity of an electrode scanned in low power-mode and proximity mode.

Table 3-1. Configuration and Status registers (continued)

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x09	1	SystemTrigger	Section 3.5.13, “System Trigger register"	0x01	RW — Configures TSS system triggering modes.
0x0A	A = number of elec.	Sensitivity	Section 3.5.14, “Sensitivity Configuration register"	0x3F	RW — Each byte represents the sensitivity for each electrode. This value is the required difference between any instant value and the baseline to indicate a touch.
0x0A + A	B = (number of elec. / 8) + 1	ElectrodeEnablers	Section 3.5.15, “Electrode enablers"	All electrodes enabled (all bytes in 0xFF)	RW — Each bit represents the enabler of an electrode. If a bit is 0, then the corresponding electrode is not scanned.
0x0A + A + B	C = (number of elec. / 8) + 1	ElectrodeStatus	Section 3.5.16, “Electrode status"	All electrodes in 0x00	R — Each bit represents the current status of an electrode. 1 is electrode detected as touched, and 0 is electrode detected as untouched.
0x0A + A + B + C	D = 2x number of electrodes	Baseline	Section 3.5.17, “Baseline register"	All electrodes in 0	R (W - after TSS_Init() call) — Each integer represents the dc-tracker baseline for each electrode.
0x0A + A + B + C + D/2	E = (number of elec. / 8) + 1	DCTrackerEnablers	Section 3.5.18, “Dc-Tracker enablers"	All electrodes enabled (all bytes in 0xFF)	RW — Each bit represents the dc-tracker enabler of an electrode. If a bit is 0, then the baseline of the corresponding electrode is not updated by the dc-tracker.
0x0A + A + B + C + D/2 + E	1	ConfigChecksum	Section 3.5.18, “Dc-Tracker enablers"	undetermined	R — This register contains the checksum value to guarantee the validity of the information contained in C&S registers.

3.5.4 Faults register

This register describes why an electrode cannot be read. Additionally, it describes other issues which may have occured in the system. When this kind of error occurs, the library stores the information about the causes of the error. The HCS08 version of the library allows enabling a software interrupt to read the Faults register and determine what caused the error. All HCS08, ColdFire V1, Coldfire+ , ARM Cortex-M0+, and ARM Cortex-M4 version support using the OnFault callback to handle the error as indicated by the Fault register.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Reserved	Reserved	Reserved	NoiseMode	SmallTriggerPeriod	Data Corruption	Small Capacitor	Charge Timeout
W								
Reset:	-	-	-	0	0	0	0	0

Figure 3-1. Faults register

Table 3-2. Faults register description

Signal	Description
NoiseMode	Indicates whether the TSS is in the noise-measurement mode and whether the Noise keydetector is active. The Noise key detector processes the noise signal from TSI module, which supports the noise mode, and is an extension of the Basic or AFID key detectors. The measurement is executed automatically in a defined rate (once during response time rate) and, if noise is detected, the keydetector is switched to the permanent noise touch detection mode. This bit is automatically cleared if the noise mode is inactive. The noise mode feature must be enabled in the TSS_SystemSetup.h. 1 — Noise mode is active 0 — Noise mode is not active
SmallTriggerPeriod	Indicates whether the trigger period is long enough to allow all active electrodes to be scanned. This bit must be cleared by writing a zero (0) to it by using the configuration function. If the SWI is enabled in the System Configuration register, an interrupt is generated. If the OnFault callback is enabled, the callback is generated. The system is disabled to prevent a cyclic error and needs to be reenabled. 1 — SmallTriggerPeriod detected 0 — No SmallTriggerPeriod detected
Data Corruption	Indicates if a change in the system configuration data is detected through an invalid direct access to memory. This bit must be cleared by writing a zero (0) to it by using the configuration function. If the SWI is enabled in the System Configuration register, an interrupt is generated. If the OnFault callback is enabled, then the callback is generated. The system is disabled to prevent a cyclic error and needs to be reenabled. If the data corruption check function is not enabled in the TSS_SystemSetup.h, the bit is not functional. For more information, see the *Touch Sensing User Guide*. 1 — Unaccounted change detected 0 — No invalid change detected

Table 3-2. Faults register description (continued)

Signal	Description
Small Capacitor	Indicates whether the small capacitance is detected on the electrode. The reason for this occurrence depends on the measurement method: • TSI method - The measured signal is below ten percent (10%) of the required TSI resolution • GPIO method - The required voltage level change is detected too fast. The measured signal is below forty (40). This bit must be cleared by writing a 0 to it by using the configuration function. If the SWI is enabled in the System Configuration register, an interrupt will be produced. If the OnFault callback is enabled, the callback is generated. The system disables the electrode(s) that caused the error. If the automatic hardware recalibration is enabled in the TSS_SystemSetup.h and there is a problem with an electrode, the hardware recalibration is executed. 1 — Small capacitor detected 0 — No small capacitor detected
Charge Timeout	Indicates whether a high capacitance is detected on the electrode. The reason for this occurrence depends on the measurement method: • TSI method - The measured signal is higher than 0xFFFF minus the 10% of the required TSI resolution • GPIO method - The required voltage level is not reached. The hardware timer timeout is exceeded. This bit must be cleared by writing a 0 to it by using the configuration function. If the SWI is enabled in the System Configuration register, an interrupt is generated. If the OnFault callback is enabled, then the callback is generated. The system disables the electrode(s) that caused this error. If the automatic hardware recalibration is enabled in the TSS_SystemSetup.h and there is a problem with an electrode, the hardware recalibration is executed. 1 — Charge timeout detected 0 — No timeout fault detected

3.5.5 System Configuration register

This register is used to enable the library features. Depending on the application needs, the features can be modified.

Register Number = 0x01

	15	14	13	12	11	10	9	8
R W	System Enabler	SWI Enabler	DC-Tracker Enabler	Stuck-key Enabler	Reserved	Reserved	Reserved	Water Tolerance Enabler
Reset	0	0	0	0	-	-	-	0

	7	6	5	4	3	2	1	0
R W	Proximity Enabler	Low Power Enabler	Reserved	Reserved	Reserved	Hardware Recalibration Starter	System Reset	Manual Recalibration Starter
Reset	0	0	-	-	-	0	0	0

Figure 3-2. System Configuration register

Table 3-3. System Configuration register descriptions

Signal	Description
System Enabler	1 — System is enabled 0 — System is disabled
SWI Enabler	If any error occurs (see the Faults register), an SWI is generated. This bit affects only the HCS08 version of the TSS library. The OnFault callback can be used for the same purpose on all HCS08, ColdfireV1, ARMCortex-M4 and ARMCortex-M0+ versions of the TSS library. 1 — SWI is generated with any fault 0 — No SWI is generated by faults
DC-Tracker Enabler	DC Tracker function enabling. The bit has higher priority than DC-tracker enablers register. 1 — The DC Tracker filtering mechanism is enabled 0 — The DC Tracker filtering mechanism is disabled
Stuck-key Enabler	1 — Stuck-key is enabled 0 — Stuck-key is disabled
Water Tolerance Enabler	1 — Water tolerance is enabled 0 — Water tolerance is disabled
Proximity Enabler	1 — Proximity Mode is enabled 0 — Proximity Mode is disabled
Low Power Enabler	1 — Low Power Mode is enabled 0 — Low Power Mode is disabled
Hardware Recalibration Starter	Writing a one (1) to this register schedules a low-level hardware calibration next time the TSS task is executed. The recalibration is performed only on electrodes which are not touched. If more electrodes belong to the same hardware module, they need to be released.
System Reset	Writing a one (1) to this register resets the system immediately
Manual Recalibration Starter	Writing a one (1) to this register schedules a baseline calibration next time the TSS task is executed

3.5.6 Number of Samples register

Register Number = 0x02

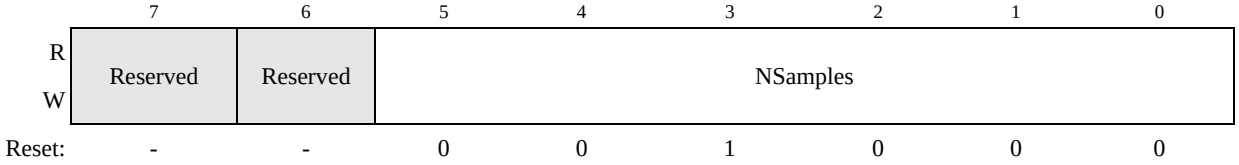

Figure 3-3. Number of Samples register

Table 3-4. Number of Samples register descriptions

Signal	Description
NSamples	If an electrode uses the GPIO method, the value determines the number of capacitance samples required to obtain a single measurement. If the electrode uses the TSI method, the value sets the NSCN register directly in the TSI module. 1–32 — *n* samples taken

3.5.7 DC Tracker Rate register

Register Number = 0x03

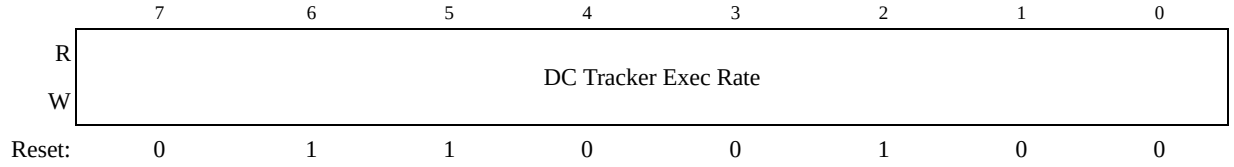

Figure 3-4. DC Tracker Rate register

Table 3-5. DC Tracker Rate register description

Signal	Description
DC Tracker Exec Rate	Determines how often the DC tracker function occurs. This number represents the number of calls to the TSS task function required before the DC tracker function is executed. If enabled, the DC tracker performs a baseline update and a negative baseline drop. If this register is set to zero (0), the DC tracker feature is disabled. 0 — DC tracker feature disabled 1–255 — DC tracker executed every *n* task executions

3.5.8 Slow DC Tracker Factor register

Register Number = 0x04

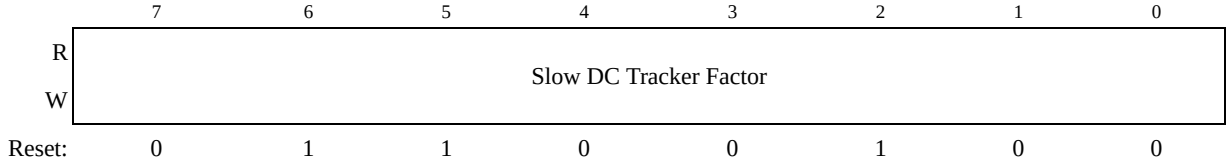

Figure 3-5. Slow DC Tracker Factor register

Table 3-6. Slow DC Tracker Factor register description

Signal	Description
Slow DC Tracker Factor	Determines the baseline updating frequency for the proximity and shielding electrodes. This number is a multiplication factor used with standard DCTrackerRate register value . 0 — Slow DC tracker feature disabled 1–255 — Slow DC tracker executed every *(n x dc_tracker_rate)* task executions

3.5.9 Response Time register

Register Number = 0x05

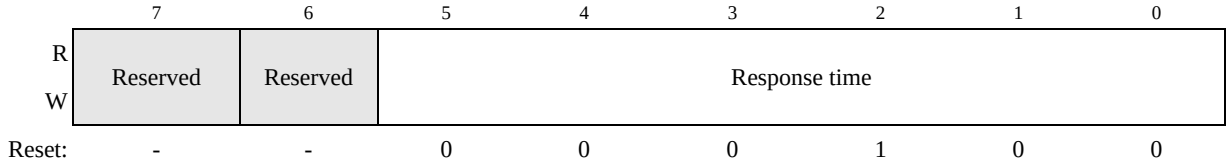

Figure 3-6. Response Time register

Table 3-7. Response Time register description

Signal	Description
Response Time	Determines the number of measurements required to perform the following: • Status change in any electrode • Single noise mode measurement to detect whether the noise is present in the system. The noise mode feature must be enabled. A single measurement of all electrodes is taken per TSS_Task execution until TSS_STATUS_OK is reported. 1–32 — *n* measurements used

3.5.10 Stuck-key Timeout register

This register configures a number of times that the library task must keep detecting a touch before it performs a forced release. The forced release is done by resetting a baseline value to a current signal value. This feature protects the application from false permanent touches caused by an external increase of the electrode capacitance. An electrode is only considered touched until the forced release is invoked. If the stuck-key function is not enabled in the TSS_SystemSetup.h, it is not possible to write to this register. For more information, see the *Touch Sense User Guide*.

Register Number = 0x06

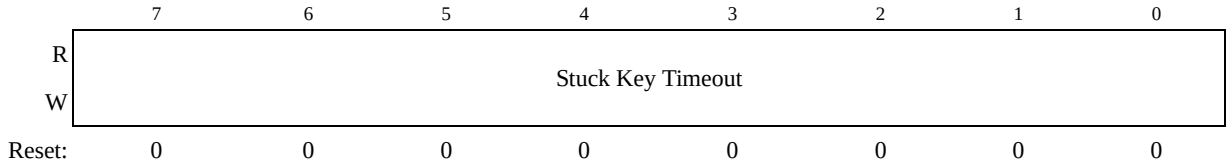

Figure 3-7. Stuck-key Timeout register

Table 3-8. Stuck-key Timeout register description

Signal	Description
Stuck Key Timeout	Determines how many task executions it takes for an electrode to be detected as touched before recalibrating that electrode and then detect that it was untouched. If this value is set to zero (0), the stuck key feature is disabled. 0 — Stuck key feature disabled 1–255 — Electrode recalibrated after n task executions.

3.5.11 Low Power Electrode register

This register represents the electrode number scanned either in a low power mode or in a proximity mode. Both in the low power mode and in the proximity mode, only one pin is active and is able to perform electrode capacitance measurements.

The low power electrode number must be selected from the module related to the low power control source defined by TSS_USE_LOWPOWER_CONTROL_SOURCE in the TSS_SystemSetup.h. If an electrode does not meet this condition, the System Setup Config Error is generated. If neither the low power control source nor the proximity callback defined in TSS_SystemSetup.h is selected, writing to this register is not possible. For more information, see the *Touch Sensing User Guide*.

If the TSI module is used as a low power control source, the register content is used to set up the TSI_PEN[LPSP].

Register Number = 0x07

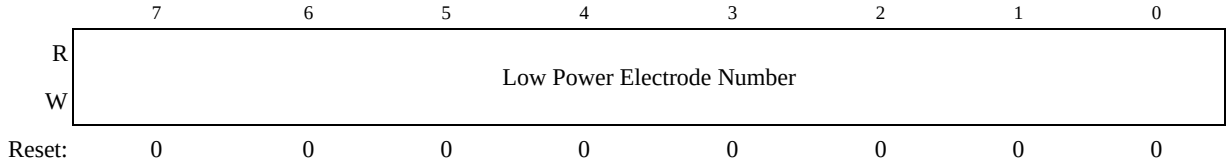

Figure 3-8. Low Power Electrode register

Table 3-9. Low Power Electrode register description

Signal	Description
Low Power Electrode Number	Number of the electrode scanned in the low power mode or in the proximity mode. The range of the electrode numbers is 0 – 63

3.5.12 Low-Power Electrode Sensitivity register

The Low-Power Sensitivity register defines the sensitivity value used to wake the system either from the low-power mode or from the sensitivity threshold for proximity detection. This sensitivity indicates the relative delta threshold value from a baseline level.

The low-power control source device manages scanning the selected electrode during the low-power mode. When the electrode value exceeds a defined value, the TSS wakes the device from the low-power mode and initializes active mode. If neither the low-power control source is selected nor the proximity callback defined in TSS_SystemSetup.h, writing to this register is not possible. See the Touch Sensing User Guide.

Register Number = 0x08

	7	6	5	4	3	2	1	0
R W	Reserved				Low Power Electrode Sensitivity			
Reset:	0	0	1	1	1	1	1	1

Figure 3-9. Low Power Electrode Sensitivity register

Table 3-10. Low Power Electrode Sensitivity register description

Signal	Description
Low Power Electrode Sensitivity	The sensitivity value of the electrode scanned in the low power mode or in the proximity mode. The range of the sensitivity is 1–127.

3.5.13 System Trigger register

This register enables and sets up library trigger features. If the trigger function is not enabled in the TSS_SystemSetup.h, writing to this register is not possible. See the *Touch Sensing User Guide* for more information.

Register Number = 0x09

	7	6	5	4	3	2	1	0
R W	Reserved	Reserved	Reserved	Reserved	Reserved	SWTrigger	TriggerMode	TriggerMode
Reset	-	-	-	-	-	0	0	1

Figure 3-10. System Trigger register

Table 3-11. System Trigger register descriptions

Signal	Description
TriggerMode	0x00 —> TSS_TRIGGER_MODE_AUTO 0x01 —> TSS_TRIGGER_MODE_ALWAYS 0x10 —> TSS_TRIGGER_MODE_SW 0x11 —> Reserved TriggerMode is not set if SWTrigger bit is set in the same moment.
SWTrigger	Write only bit. This bit controls software triggering scan execution. In case the TSS_TRIGGER_MODE_SW is selected, write 1 to the software trigger performs one scan cycle execution, otherwise the bit is not functional.

3.5.14 Sensitivity Configuration register

This set of registers contains a sensitivity value for each electrode. The sensitivity value is an 8-bit value that represents the minimum difference between an instant signal value and the baseline, which is required to indicate a touch. The sensitivity does not need to be set if Automatic Sensitivity Calibration is enabled. For more information, see the *Touch Sensing User Guide*.

Register Number = 0x0A

	7	6	5	4	3	2	1	0
R W		Sensitivity						
Reset:	-	0	1	1	1	1	1	1

Figure 3-11. Sensitivity Configuration register

Table 3-12. Sensitivity Configuration register description

Signal	Description
Sensitivity	When an electrode is detected as touched, represents the threshold in a signal delta. 2–127 — Electrode sensitivity

3.5.15 Electrode enablers

This set of registers inform which electrode is enabled. Each bit represents one electrode where a bit value one (1) represents an enabled electrode, and a bit value zero (0) represents a disabled electrode.

All dc-tracker enabler bits are enabled after reset.

Because certain errors automatically disable electrodes, the application must re-enable them. If an automatic hardware recalibration is enabled in the TSS_SystemSetup.h, the hardware recalibration bit is automatically set in the System Config register which results in hardware recalibration whenever the next TSS_Task() is executed.

Note that enabling or disabling controls also enables or disables all electrodes assigned to the control. However, setting the register manually is always allowed.

The arrangement of electrodes with bits is as follows:

- Register 0, bit 0 — Electrode 0
- Register 0, bit 1 — Electrode 1
- Register 0, bit 7 — Electrode 7
- Register 1, bit 0 — Electrode 8
- Register 1, bit 7 — Electrode 15
- etc.

If the electrode enabler settings are changed, a hardware recalibration is scheduled to configure the system to the new electrode configurations. The hardware recalibration starter bit in the System Configuration register is automatically set for this purpose. For more information, see Section 3.5.5, “System Configuration register”.

3.5.16 Electrode status

This set of registers provides the electrode status (touched or untouched). Each bit represents one electrode. A value one (1) represents a touched electrode, while a value (0) represents an untouched electrode.

The arrangement of electrodes with bits is as follows:

- Register 0, bit 0 — Electrode 0
- Register 0, bit 1 — Electrode 1
- Register 0, bit 7 — Electrode 7
- Register 1, bit 0 — Electrode 8
- Register 1, bit 7 — Electrode 15
- etc.

3.5.17 Baseline register

This register is used to read and write dc-tracker baseline values. Writing is allowed only after TSS_Init() call and before the first TSS_Task() execution.

Register Number = 0x0A + A + B + C

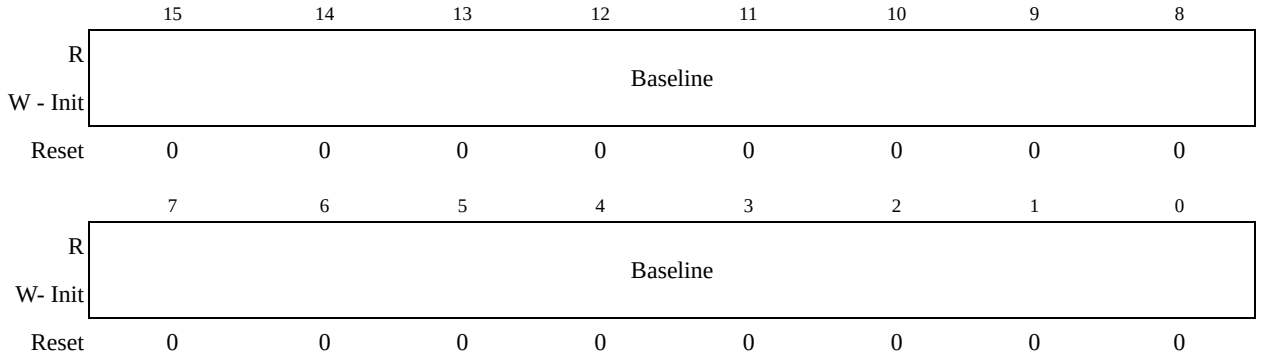

Figure 3-12. Baseline register

Table 3-13. Baseline register descriptions

Signal	Description
Baseline	Represents the dc-tracker baseline value. Writing is allowed only after TSS_Init() call and before the first TSS_Task() execution. 0–65535 — Electrode baseline

3.5.18 Dc-Tracker enablers

This set of registers detect which electrode has an enabled update of the dc-tracker baseline. Each bit represents one electrode. Bit value one (1) represents an enabled dc-tracker on an electrode, while a value (0) represents a disabled dc-tracker on an electrode. All dc-tracker enabler bits are enabled after reset.

If one control electrode is touched, analog decoders automatically disable a dc-tracker on all electrodes assigned to the control . If all electrodes from the control are released, the dc-tracker automatically re-enables these electrodes. Setting the register manually is also an option.

The Dc-tracker enabler bit in the System Configuration register is a global enabler of the dc-tracker function with a higher priority than the dc-tracker enablers.

The arrangement of dc-tracker enablers with bits is as follows:

- Register 0, bit 0 — Dc-tracker on electrode 0
- Register 0, bit 1 — Dc-tracker on electrode 1
- Register 0, bit 7 — Dc-tracker on electrode 7
- Register 1, bit 0 — Dc-tracker on electrode 8
- Register 1, bit 7 — Dc-tracker on electrode 15
- etc

3.5.19 Configuration Checksum Register

This read-only register contains the checksum value to guarantee the validity of the information contained in C&S registers. When the TSS_SetSystemConfig() function is called to update a C&S register, a

checksum algorithm is performed and the result is stored in this register. The TSS_Task() function checks the integrity of this checksum. If it fails, the data corruption bit of the fault register is set. This function can be disabled in the TSS_SystemSetup.h by defining the TSS_USE_DATA_CORRUPTION_CHECK. For more information, see the *Touch Sensing User Guide*.

NOTE

If you write to the Configuration and Status registers directly, without using the TSS_SetSystemConfig() function, the Checksum register will not be updated and the data corruption bit will be set on the next execution of the TSS_Task() function.

3.6 Keypad decoder API

This section describes the keypad decoder API. It describes the keypad Configuration and Status registers, the TSS_SetKeypadConfig() function used to write to these registers, and the data structure used for reading this register. This section also describes the callback function and its parameters for the keypad control.

Each application keypad control has its respective Configuration and Status register and a callback function.

3.6.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, use the TSS_SetKeypadConfig function declared in the TSS_API.h file.

Function prototype

```
uint8_t TSS_SetKeypadConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter, uint8_t
u8Value);
```

Input parameters

Return valueReturn value

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control's C&S structure
uint8_t	u8Parameter	Any of the parameter codes provided by keypad decoder	Code indicating the parameter configured
uint8_t	u8Value	Depends on the specific parameter configured	New desired value, for the respective configuration register

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	Configuration is successful.
TSS_ERROR_KEYPAD_ILLEGAL_CONTROL_TYPE	Configuration is not executed. The Id parameter does not match the control type structure that the user was trying to modify.
TSS_ERROR_KEYPAD_READ_ONLY_PARAMETER	Configuration is not executed whenever the user attempts to modify a read-only parameter.
TSS_ERROR_KEYPAD_ILLEGAL_VALUE	Configuration is not executed because of an illegal value number.
TSS_ERROR_KEYPAD_OUT_OF_RANGE	Configuration is not executed because the new value was out of the established boundaries.
TSS_ERROR_KEYPAD_ILLEGAL_PARAMETER	Configuration is not executed because of an illegal parameter number.

3.6.2 Reading the Configuration and Status registers

The two options to read the keypad Configuration and Status register structure involve either using the TSS function declared in the TSS_API.h file or directly accessing the configuration and status structure for specific data. The following describes the first option.

Function prototype

```
uint8_t TSS_GetKeypadConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter);
```

Input parameters

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control's C&S structure
uint8_t	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds either to an actual unsigned byte value of a register specified by u8Parameter, or error values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned byte value	Value corresponds to an actual unsigned byte value of a register specified by u8Parameter.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Read is not successful because the parameter is illegal.
TSS_ERROR_GETCONF_ILLEGAL_CONTROL_TYPE	Read is not successful because the control type is illegal.

The second option to read the keypad Configuration and Status register structure involves directly accessing the configuration and status structure for specific data. The structure name is defined by the application in the TSS_SystemSetup.h file by using the following definition:

```
#define TSS_Cn_STRUCTURE cKey0
```

Where *n* is the Control number starting from zero (0) to the number of controls minus one.

The cKey0 name is just an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_KEYPAD_CONTCONF ControlConfig;                 //Note 2
        uint8_t BufferReadIndex;
        const uint8_t BufferWriteIndex;
        const TSS_KEYPAD_EVENTS Events;                          //Note 3
        const uint8_t MaxTouches;
        const uint8_t AutoRepeatRate;
        const uint8_t AutoRepeatStart;
        const uint8_t * const BufferPtr;
} TSS_CSKeypad;
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.6.4, "Control ID register"”

2. The TSS_KEYPAD_CONTCONF type is an eight bit-field structure described in Section 3.6.5, “Control Configuration register"”
3. The TSS_KEYPAD_EVENTS type is an eight bit-field structure described in Section 3.6.9, “Event Control and Status register"”

The following command shows how to read the Control ID register by using the example of the structure *cKey0*:

```
Temp = cKey0.ControlId;
```

3.6.3 Configuration and Status registers list

The following table describes the keypad control Configuration and Status registers, and the Control ID Register.

Table 3-14. Keypad Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.6.4, “Control ID register"”	Application dependable	R — Displays the control type and control number.
0x01	1	ControlConfig	Section 3.6.5, “Control Configuration register"”	0x00	RW — Configures overall enablers of the object.
0x02	1	BufferReadIndex	Section 3.6.7, “BufferReadIndex"”	0x00	RW — Index of the first unread element of the events buffer
0x03	1	BufferWriteIndex	Section 3.6.8, “BufferWriteIndex"”	0x00	R — Index of the first free element of the buffer
0x04	1	Event Control and Status Register	Section 3.6.9, “Event Control and Status register"”	0x00	RW — Configures the events that will be reported by the controller. This Register holds the Max Key and Buffer Overflow Flags
0x05	1	MaxTouches	Section 3.6.10, “MaxTouches register"”	0x00	RW — Configures the maximum number of keys that can be pressed at the same time.
0x06	1	AutoRepeatRate	Section 3.6.11, “Auto Repeat Rate register"”	0x00	RW — Configures the rate at which keys will be reported when they are kept pressed and no movement is detected.

Table 3-14. Keypad Control Configuration and Status registers (continued)

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x07	1	AutoRepeatStart	Section 3.6.12, “Auto Repeat Start register"”	0x00	RW — Configures the time between a touch and an auto-repeat event when the key is kept pressed.
0x08	1	BufferPtr	Section 3.6.6, “Buffer Pointer register"”	Assigned at precompile time	R — Address of the buffer where the events for each controller are stored

3.6.4 Control ID register

This read-only register contains the identifier code of the control.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-13. Control ID Register

Encoding	Control Type
001	Keypad
010	Slider
011	Rotary
100	Analog slider
101	Analog rotary
110	Matrix
100-111	Reserved

NOTE

The control number is assigned in the system setup module. This value is unique for all controls, indicating that each control has a particular number regardless of its type. The first assigned control number is zero.

3.6.5 Control Configuration register

This register configures the overall behavior of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R W	Control Enabler	Callback Enabler	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Reset:	0	0	-	-	-	-	-	-

Figure 3-14. Control Configuration register

Table 3-15. Control Configuration register description

Signal	Description
Control Enabler	Global enabler for the control. This determines if the control electrodes are scanned for touch detection. The bit either enables or disables all control electrodes in the Electrode Enablers register. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function will be called when one of the active events in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled

3.6.6 Buffer Pointer register

This register defines the base address of the event buffer for a specific control. The event buffer is a circular buffer with 16 elements. It stores every touch or release event that occurs in the control depending on the events enabled in the Events Register.

The Keypad Events Register has the following 8-bit format:

Register Number = 0x02

	7	6	5	4	3	2	1	0
R	Event Type	Reserved	Reserved	Reserved	Key Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Assigned at TSS_Init

Figure 3-15. Buffer Pointer register

Table 3-16. Buffer Pointer register description

Signal	Description
Event Type	Indicates the type of event registered in the buffer. 1 — Release event 0 – Touch event
Key Number	Determines the key within keypad control on which the event has occurred. 0–15 — Key presenting the event

3.6.7 BufferReadIndex

This register stores the index of the first unread element of the events buffer. This index is used to start reading the data from the buffer until it reaches the value of the Buffer Write Index. The user is responsible for updating this value after the read operation, which can be done automatically by using the TSS_KEYPAD_BUFFER_READ macro.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R W	Reserved	Reserved	Reserved	Reserved	Index value			
Reset:	-	-	-	-	0	0	0	0

Figure 3-16. BufferReadIndex register

Table 3-17. BufferReadIndex Register description

Signal	Description
Index value	Determines the index of the first unread event in the events buffer. 0–15 — Index of first unread element

Macro prototype

```
#define TSS_KEYPAD_BUFFER_READ(destvar,kpcsStruct)
```

This macro allows you to store the first unread element from the event buffer and update the value of the Buffer Write Index register. When using this macro, provide two parameters: destvar and kpcsStruct.

Input parameters

Type	Name	Description
uint8_t	destvar	Name of the variable where the first unread element is stored.
CSKeypad	kpcsStruct	Name of the controller structure defined in the TSS_SystemSetup.h header file. See the *Touch Sensing User Guide* for more details about the structure name.

Return value

None

3.6.8 BufferWriteIndex

This register stores the index value of the first free element in the buffer which is located one element after the last captured event. This register is automatically updated when new events are stored in the buffer and is provided for reference during the read operations. The user should read the buffer until this index is reached. This section also describes the TSS_KEYPAD_BUFFER_EMPTY macro.

Register Number = 0x04

	7	6	5	4	3	2	1	0
R	Reserved	Reserved	Reserved	Reserved	Index value			
W								
Reset:	-	-	-	-	0	0	0	0

Figure 3-17. BufferWriteIndex register

Table 3-18. BufferWriteIndex register description

Signal	Description
Index value	Determines the index of the first free element in the events buffer. 0–15 — Index of the first free element

Macro prototype

```
#define TSS_KEYPAD_BUFFER_EMPTY(kpcsStruct)
```

This macro indicates when the events buffer is empty. The macro performs a comparison between the Buffer Read Index and the Buffer Write Index.

Input parameters

Type	Name	Description
CSKeypad	kpcsStruct	Name of the controller structure defined by the user in the TSS_SystemSetup.h header file. See the *Touch Sensing User Guide* for more details about the structure name.

Return value

The macro performs a comparison between the first unread element index and the first free element index in the buffer. It indicates that all elements in the buffer have been read when the two indexes are equal. If all elements have been read, the macro returns one (1), if not, it returns zero (0).

3.6.9 Event Control and Status register

This register contains bits used to enable or disable events that call the control callback function. It also contains the maximum keys and buffer overflow status flags.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R	Max Keys Flag	Buffer Overflow Flag	Reserved	Keys Exceeded Enabler	Buffer Full/Overflow Enabler	Auto-Repeat Enabler	Release Event Enabler	Touch Event Enabler
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-18. Event Control and Status register

Table 3-19. Event Control and Status register description

Signal	Description
Max Keys Fault	A flag that indicates whether the limit of keys pressed at the same time has been exceeded. The limit is defined by the MaxTouches Register. In this case, any further key touches will be ignored until at least one key is released. This flag will be automatically cleared after the fault condition turns false. 1 — Limit exceeded 0 — No excess keys
Buffer Overflow Flag	This flag indicates whether more events have been produced than the buffer free space allows. In this case the newest events are lost and no more events can be logged. This flag is cleared by writing a zero (0) to this bit. 1 — Buffer overflowed 0 — Free space available
Keys Exceeded Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function after the limit for pressed keys is exceeded. 1 — Event enabled 0 — Event disabled
Buffer Overflow Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function after the circular buffer for events is full or overflown. 1 — Event enabled 0 — Event disabled
Auto-repeat Enabler	If this bit is set, the decoder stores a new touch event in the events buffer at the specified rate in case at least one key remains pressed for the time configured in the Auto-repeat Start register. If the control's callback function is enabled, it is called at the same rate. 1 — Auto-repeat enabled 0 — Auto-repeat disabled
Release Event Enabler	If this bit is set, the decoder stores released events in the events buffer. If the control callback function is enabled, it is called when an event occurs. 1 — Event enabled 0 — Event disabled
Touch Event Enabler	If this bit is set, the decoder stores touch events in the events buffer. If the control callback function is enabled, it is called when an event occurs. 1 — Event enabled 0 — Event disabled

NOTE

If touch and release events are both disabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.6.10 MaxTouches register

The MaxTouches register defines the maximum number of keys reported per touch. This feature controls the function when a larger area than expected is touched. For example, a multiplexed array of electrodes where only two keys are needed to detect a value or a function. In this case, the MaxTouches register is configured with a value of two.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Reserved	Reserved	Reserved	Reserved	Max Number of Touches			
Reset:	-	-	-	-	0	0	0	0

Figure 3-19. MaxTouches register

Table 3-20. MaxTouches register description

Signal	Description
Max Number of Touches	Sets the limit for simultaneous active keys. If set to zero (0), no limit is established. 0 — No limit established 1–15 — Limit set to *n* keys

3.6.11 Auto Repeat Rate register

This register contains the rate value at which the pressed keys are reported when kept pressed.

Register Number = 0x07

	7	6	5	4	3	2	1	0
R W	Auto-repeat rate							
Reset:	0	0	0	0	0	0	0	0

Figure 3-20. Auto Repeat Rate register

Figure 3-21. Auto Repeat Rate register description

Signal	Description
Auto-repeat Rate	Sets the rate where the pressed keys are reported when kept pressed. This value is a multiplier of the touch sensing task execution rate. If set to zero (0), no auto-repeat occurs and the event is automatically disabled in the Events Register. The user must manually re-enable this event if desired. 0 — Auto-repeat feature disabled 1–255 — Touch event reported every *n* execution times

3.6.12 Auto Repeat Start register

This register defines the time difference between a touch event and its auto-repeat, which is ensuring that the key remains touched. During this time, no event is generated. After this time elapses, a new event is generated and the auto-repeat rate register controls the new event timing by generating new touch events at a specified rate. After this, the value of this register is not used by the decoder until a different touch event is detected.

Register Number = 0x08

	7	6	5	4	3	2	1	0
R W	Auto-repeat Start							
Reset:	0	0	0	0	0	0	0	0

Figure 3-22. Auto Repeat Start register

Table 3-21. Auto Repeat Start register description

Signal	Description
Auto-repeat Start	Sets the length a key must remain pressed before generating new events at the auto-repeat rate. This value is a multiplier of the touch sensing task execution rate. If set to 0, the auto-repeat state will be entered immediately after the last touch. 0–255 — Wait *n* execution times before auto-repeat.

3.6.13 Keypad Callback function

This function is called by the decoder module if an event occurs and the callback function is enabled. Callback functions are assigned to controls in the system setup module and one callback may be assigned to different controls in the system.

Function prototype

```
void CallbackFuncName(uint8_t u8ControlId)
```

CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK fCallBack1
```

Where:

- *n* is the number of the control
- fCallBack1 is a name example

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8ControlId	Any valid control ID of any control in the system.	It indicates the control that generated the event. This parameter matches the controlled field in the control C&S register.

Return Value

None

3.7 Slider and Rotary decoder API

This section describes the API for slider and rotary decoder. Because the rotary decoder API is similar to the slider API, this section describes the slider decoder and the differences between the two. The

TSS_SetSliderConfig() and TSS_SetRotaryConfig() function are used to write to the decoder registers. The TSS_GetSliderConfig() and TSS_GetRotaryConfig() are used for reading the registers. This section describes the callback function and its parameters for a slider and rotary controls. Each application slider and rotary control has its respective Configuration and Status registers and a callback function.

3.7.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status registers, call TSS_SetSystemConfig function. This is declared in the TSS_API.h file.

Function prototype for Slider

```
uint8_t TSS_SetSliderConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter, uint8_t
u8Value);
```

Function prototype for Rotary

```
uint8_t TSS_SetRotaryConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter, uint8_t
u8Value);
```

Input parameters

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured which is stored in the first element of the control's C&S structure.
uint8_t	u8Parameter	Any of the parameter codes provided by slider decoder	Code indicating that the parameter is configured.
uint8_t	u8Value	Depends on the specific parameter configured	New desired value for the respective configuration registers

Return value for Slider

The return value is an unsigned integer with the following possible return values defined in the file TSS_API.h:

Return Value	Description.
TSS_STATUS_OK	Configuration is successful.
TSS_ERROR_ SLIDER _ILLEGAL_PARAMETER	Configuration is not executed because of an illegal parameter number.
TSS_ERROR_ SLIDER _READ_ONLY_PARAMETER	Configuration is not executed whenever the user attempts to modify a read-only parameter.
TSS_ERROR_ SLIDER _OUT_OF_RANGE	Configuration is not executed because the new value is out of range of the established boundaries.
TSS_ERROR_ SLIDER _ILLEGAL_CONTROL_TYPE	Configuration is not executed because the u8ControlId parameter does not match the control type structure that the user was trying to modify.

Return value for Rotary

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration is successful.
TSS_ERROR_ROTARY_ILLEGAL_PARAMETER	Configuration is not executed because of the illegal parameter number.
TSS_ERROR_ROTARY_READ_ONLY_PARAMETER	Configuration is not executed whenever the user attempts to modify a read-only parameter.
TSS_ERROR_ROTARY_ILLEGAL_VALUE	Configuration is not executed because of an illegal value number.
TSS_ERROR_ROTARY_OUT_OF_RANGE	Configuration is not executed because the new value was out of the established boundaries
TSS_ERROR_ROTARY_ILLEGAL_CONTROL_TYPE	Configuration is not succesful because the u8ControlId parameter does not match the control type structure that the user is trying to modify.

3.7.2 Reading the Configuration and Status registers

The two options to access Configuration and Status register structure involve using theTSS function declared in the TSS_API.h file and directly accessing the configuration and status structure for specific data. This following content describes the first option.

Function prototype for Slider

```
uint8_t TSS_GetSliderConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter);
```

Function prototype for Rotary

```
uint8_t TSS_GetRotaryConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter);
```

Input parameters

Return value

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured which is stored in the first element of the control's C&S structure
uint8_t	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds to either a register unsigned byte value specified by the u8Parameter, or error values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned byte value	Value corresponds to an unsigned byte value of a register specified by u8Parameter.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Read was not successful because of the illegal parameter.
TSS_ERROR_GETCONF_ILLEGAL_CONTROL_TYPE	Read was not successful because of the illegal control type.

The second option to access Configuration and Status register structure involves direct access to the configuration and status structure for specific data. The application defines the structure name in the TSS_SystemSetup.h file by using the following definition:

```
#define TSS_Cn_STRUCTURE      cStructure
```

Where *n* is the Control number starting from 0 going up to the number of controls minus one.

The *cStructure* name is just an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_SLIDER_CONTROL ControlConfig;                  //Note 2
        const TSS_SLIDER_DYN DynamicStatus;                      //Note 3
        const TSS_SLIDER_STAT StaticStatus;                      //Note 4
        const TSS_SLIDER_EVENTS Events;                          //Note 5
        const uint8_t AutoRepeatRate;
        const uint8_t MovementTimeout;
} TSS_CSSlider;

typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_SLIDER_CONTROL ControlConfig;                  //Note 2
        const TSS_SLIDER_DYN DynamicStatus;                      //Note 3
        const TSS_SLIDER_STAT StaticStatus;                      //Note 4
        const TSS_SLIDER_EVENTS Events;                          //Note 5
        const uint8_t AutoRepeatRate;
        const uint8_t MovementTimeout;
} TSS_CSRotary;
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.7.4, “Control ID register"”
2. The TSS_SLIDER_CONTROL type is an eight bit-field structure described in Section 3.7.5, “Control configuration"”
3. The TSS_SLIDER_DYN type is an eight bit-field structure described in Section 3.7.6, “Dynamic Status register"”
4. The TSS_SLIDER_STAT type is an eight bit-field structure described in Section 3.7.7, “Static Status register"”
5. The SLIDER_EVENTS type is an eight bit-field structure described in Section 3.7.8, “Events Control register"”

The following command is an example command used to read the Control ID register:

```
Temp = cStructure.ControlId;
```

3.7.3 Configuration and Status registers list

The following table contains the control Configuration and Status registers.

Table 3-22. Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.7.4, “Control ID register””	Application dependable	R — Displays the control type and control number.
0x01	1	ControlConfig	Section 3.7.5, “Control configuration””	0x00	RW — This register configures overall enablers of the object.
0x02	1	DynamicStatus	Section 3.7.6, “Dynamic Status register””	0x00	R — Displays movement, direction and displacement information.
0x03	1	StaticStatus	Section 3.7.7, “Static Status register””	0x00	R — Displays touch and absolute position information. Hold the invalid position status flag.
0x04	1	Events	Section 3.7.8, “Events Control register””	0x00	RW — Configures the events that call the callback function.
0x05	1	AutoRepeatRate	Section 3.7.9, “Auto-repeat Rate register””	0x00	RW — Configures the rate at which keys are reported when they are kept pressed and no movement is detected.
0x06	1	MovementTimeout	Section 3.7.10, “Movement Timeout register””	0x00	RW — Number of times the decoder must detect a no displacement before reporting a hold event and no movement.

3.7.4 Control ID register

This read-only register contains the control identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-23. Control ID register

Encoding	Control Type
001	Keypad
010	Slider
011	Rotary
100	Analog Slider
101	Analog Rotary
110	Matrix
111	Reserved

NOTE

The control number is assigned in the system setup module. This value is unique for all controls and indicates that each control has a particular number regardless of its type. The first assigned control number is zero.

3.7.5 Control configuration

This register configures overall enablers of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R W	Control Enabler	Callback Enabler	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Reset:	0	0	-	-	-	-	-	-

Figure 3-24. Control Configuration register

Table 3-23. Control Configuration register description

Signal	Description
Control Enabler	Global enabler for the control. This determines if the electrodes of the control are scanned for detection. The bit enables or disables all control electrodes in the Electrode enablers register. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function is called whenever an active event occurs in the Events Configuration registers. 1 — Callback enabled 0 — Callback disabled

3.7.6 Dynamic Status register

This register describes the movement of the control over the electrodes.

Register Number = 0x02

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	0	0	Displacement			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-25. Dynamic Status register

Table 3-24. Dynamic Status register description

Signal	Description
Movement Flag	Indicates whether movement is detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of the movement. This bit remains with its last value even if movement stops and is no longer detected. 1 — Incremental (from Ex to Ey, where x < y) 0 — Decremental (from Ex to Ey, where x > y)
Displacement	This value indicates the difference in positions from the last status to the new one. This indicates how many positions have been advanced in the current movement direction. 0–15 — Number of positions.

3.7.7 Static Status register

This register displays events, flags, and status of the electrodes in the control when no movement over the electrodes is detected.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R	Touch Flag	Invalid Position Flag	0	Position				
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-26. Static Status register

Table 3-25. Static Status register description

Signal	Description
Touch Flag	Indicates whether a touch is detected in the control at the moment of reading. 1 — Touch detected 0 — Touch not detected

Signal	Description
Invalid Position Flag	Indicates if an invalid combination of touched electrodes is being detected. 1 — Invalid position detected 0 — Valid position detected
Position	This value indicates the absolute position within the control that is actuated. 0–31 — Absolute position

3.7.8 Events Control register

The register contains bits used to enable or disable events that call the control callback function.

Register Number = 0x04

	7	6	5	4	3	2	1	0
R W	Reserved	Reserved	Reserved	Release Event Enabler	Hold Auto-Repeat Enabler	Hold Event Enabler	Movement Event Enabler	Initial Touch Event Enabler
Reset:	0	0	0	0	0	0	0	0

Figure 3-27. Events Control register

Table 3-26. Events Control Register description

Signal	Description
Release Event Enabler	If this bit is set and the callback function is enabled, the callback is executed when all touched electrodes in the control are released. 1 — Release event enabled 0 — Release event disabled
Hold Auto-repeat Enabler	If this bit is set, the Hold Event Enabler and the callback function are enabled. The callback is executed at the rate specified in the Auto-repeat Rate register for as long as one valid position is detected as touched in the control, and no movement is detected. 1 — Auto-repeat feature enabled 0 — Auto-repeat disabled
Hold Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function when the movement stops and, after a certain period of time, a constant touched position is detected. The time is configurable in the Movement Timeout register. 1 — Event enabled 0 — Event disabled
Movement Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function every time a displacement is detected. 1 — Event enabled 0 — Event disabled
Initial Touch Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function when the control transitions from an idle to an active state. 1 — Event enabled 0 — Event disabled

NOTE

If none of the events are enabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.7.9 Auto-repeat Rate register

The Auto-repeat Rate register contains the rate value where a hold event is reported when it is kept pressed without movement.

Register Number = 0x05

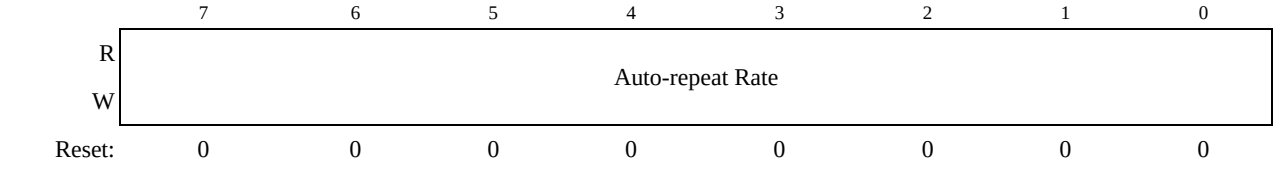

Figure 3-28. Auto-repeat Rate register

Table 3-27. Auto-repeat Rate register description

Signal	Description
Auto-repeat Rate	The decoder calls the control callback function at a specified rate provided that a valid position is continuously detected, the auto-repeat feature is enabled and no displacement occurs. If this register is set to 0, the auto-repeat feature will be disabled in the events register. 0 — Auto-repeat feature disabled 1–254 — Callback is called every *n* task executions

NOTE

The auto-repeat function works only if a hold event is presented. The hold event is delayed by the Movement Timeout function. If the Movement Timeout register is set to zero (0), the Auto-repeat function stops to generate callback because the Hold event is no longer reported.

3.7.10 Movement Timeout register

The register value defines how long the movement event stays reported after the hold event is detected. The delay is defined in number of `TSS_Task()` executions. The hold indicates the instance when the movement stops and a constant touched position is detected.

Register Number = 0x06

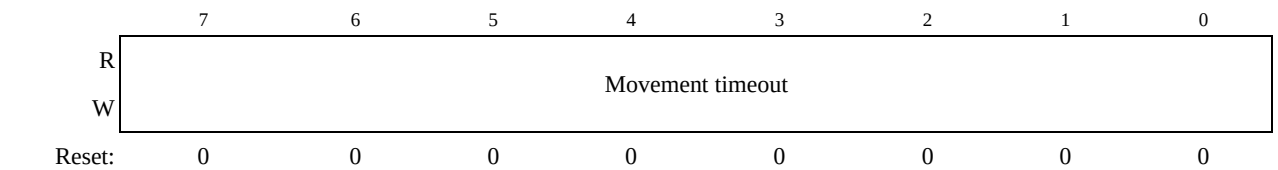

Figure 3-29. Movement Timeout register

Table 3-28. Movement Timeout register description

Signal	Description
Movement Timeout	Determines how many execution times the decoder must detect that there is no displacement before reporting no movement and a hold event. 0 — No limit established. Movement is reported until the control is detected as idle. 1–254 — Reports a hold event after *n* task executions.

3.7.11 Callback function

The decoder module calls this function if an event occurs and the user enables the callback function. Callback functions are assigned to controls in the system setup module. One callback may be assigned to several different controls in the system.

Function prototype

```
void CallbackFuncName(uint8_t u8ControlId)
```

The CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK fCallBack3  /* Identifier of the user's callback */
```

Where:

- *n* is the number of the control
- *fCallBack3* is a name example

Input parameters

Type	Name	Valid Range/Values	Description
uint8_t	u8ControlId	Any valid Control ID of any control in the system.	This parameter indicates which control generates the event. This parameter matches the controlled field in the control C&S structure.

Return value

None

3.8 Analog slider and analog rotary decoder API

This section describes the API for the analog slider and the analog rotary decoder. Because the analog rotary decoder API is similar to the analog slider, this section describes both the analog slider decoder and differences between the two. The TSS_SetASliderConfig() and TSS_ASetRotaryConfig() function are used to write to the decoder registers. The TSS_GetASliderConfig() and TSS_GetARotaryConfig() are used for reading the registers. This section describes the callback function and its parameters for an analog slider and a analog rotary control. Each application analog slider and an analog rotary control has the corresponding Configuration and Status registers and a callback function.

3.8.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, call the TSS_SetSystemConfig function. This function is declared in the TSS_API.h file.

Function prototype for Analog Slider

```
uint8_t TSS_SetASliderConfig (TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter, uint8_t
u8Value)
```

Function prototype for Analog Rotary

```
uint8_t TSS_SetARotaryConfig (TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter, uint8_t
u8Value)
```

Input parameters

Type	Name	Valid range and values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured which is stored in the first element of the control's C&S structure.
uint8_t	u8Parameter	Any of the parameter codes provided by the decoder	Code indicating the parameter configured
uint8_t	u8Value	Depends on the specific parameter configured	New desired value, for the respective configuration registers

Return value for Analog Slider

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration is executed successfully.
TSS_ERROR_ASLIDER_ILLEGAL_PARAMETER	Configuration is not executed because of the illegal parameter number.
TSS_ERROR_ASLIDER_ILLEGAL_VALUE	Configuration is not executed because of the illegal value.
TSS_ERROR_ASLIDER_READ_ONLY_PARAMETER	Configuration is not executed whenever the user attempts to modify a read-only parameter.
TSS_ERROR_ASLIDER_ILLEGAL_VALUE	Configuration is not executed because of the illegal value number.
TSS_ERROR_ASLIDER_OUT_OF_RANGE	Configuration is not executed because the new value was out of range of the established boundaries.
TSS_ERROR_ASLIDER_ILLEGAL_CONTROL_TYPE	Configuration is not executed because the u8ControlId parameter does not match the control type structure that the user is trying to modify.

Return value for Analog Rotary

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration is executed successfully.
TSS_ERROR_AROTARY_ILLEGAL_PARAMETER	Configuration is not executed because of the illegal parameter number.
TSS_ERROR_AROTARY_ILLEGAL_VALUE	Configuration is not executed because of the illegal value.
TSS_ERROR_AROTARY_READ_ONLY_PARAMETER	Configuration is not executed whenever the user attempts to modify a read-only parameter.
TSS_ERROR_AROTARY_ILLEGAL_VALUE	Configuration is not executed because of the illegal value number.
TSS_ERROR_AROTARY_OUT_OF_RANGE	Configuration is not executed because the new value is out of range of the established boundaries.
TSS_ERROR_AROTARY_ILLEGAL_CONTROL_TYPE	Configuration is not executed because the u8ControlId parameter does not match the control type structure that the user is trying to modify.

3.8.2 Reading the Configuration and Status registers

The two options to access the rotary Configuration and Status register structure involve using the TSS function declared in the TSS_API.h file and directly accessing the configuration and status structure for specific data. The following content describes the first option.

Function prototype for Analog Slider

```
uint8_t TSS_GetASliderConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter);
```

Function prototype for Analog Rotary

```
uint8_t TSS_GetARotaryConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter);
```

Input parameters

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured which is stored in the first element of the control C&S structure.
uint8_t	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds either to a control register unsigned byte value specified by the u8Parameter, or to error values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned byte value	Value corresponds to an unsigned byte value of a register specified by the u8Parameter.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Read is not successful because of the illegal parameter.
TSS_ERROR_GETCONF_ILLEGAL_CONTROL_TYPE	Read is not successful because of the illegal control type.

The second option enables direct access to the configuration and status structure for specific data. The application defines the structure name in the TSS_SystemSetup.h file by using the following definition:

```
#define TSS_Cn_STRUCTURE      cStructure
```

Where *n* is the control number starting from 0 going up to the number of controls minus one.

The *cStructure* name is an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_ASLIDER_CONTROL ControlConfig;                 //Note 2
        const TSS_ASLIDER_DYN DynamicStatus;                     //Note 3
        const uint8_t Position;
        const TSS_ASLIDER_EVENTS Events;                         //Note 4
        const uint8_t AutoRepeatRate;
        const uint8_t MovementTimeout;
        const uint8_t Range;
} TSS_CSASlider;

typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_ASLIDER_CONTROL ControlConfig;                 //Note 2
        const TSS_ASLIDER_DYN DynamicStatus;                     //Note 3
        const uint8_t Position;
        const TSS_ASLIDER_EVENTS Events;                         //Note 4
        const uint8_t AutoRepeatRate;
```

```
        const uint8_t MovementTimeout;
        const uint8_t Range;
} TSS_CSARotary;
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.8.4, “Control ID register".”
2. The TSS_ASLIDER_CONTROL type is an eight bit-field structure described in Section 3.8.5, “Control configuration".”
3. The TSS_ASLIDER_DYN type is an eight bit-field structure described in Section 3.8.6, “Dynamic Status register"”
4. The TSS_ASLIDER_EVENTS type is an eight bit-field structure described in Section 3.8.8, “Events Control register"”

The following command is an example to read the Control ID:

```
Temp = cStructure.ControlId;
```

3.8.3 Configuration and Status registers list

The following table shows the control Configuration and Status registers.

Table 3-29. Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.8.4, “Control ID register"”	Application dependable	R — Displays the control type and control number
0x01	1	ControlConfig	Section 3.8.5, “Control configuration"”	0x00	RW — This register configures overall enablers of the object
0x02	1	DynamicStatus	Section 3.8.6, “Dynamic Status register"”	0x00	R — Displays movement, direction, and displacement information
0x03	1	Position	Section 3.8.7, “Position register"”	0x00	R — Displays absolute position information within a defined range.
0x04	1	Events	Section 3.8.8, “Events Control register"”	0x00	RW — Configures the events that execute the callback function.
0x05	1	AutoRepeatRate	Section 3.8.9, “Auto-repeat Rate register"”	0x00	RW — Configures the rate at which keys are reported when they are kept pressed and no movement is detected.

Table 3-29. Control Configuration and Status registers (continued)

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x06	1	MovementTimeout	Section 3.8.10, “Movement Timeout register"”	0x00	RW — Number of times the decoder must detect a no-displacement before reporting a hold event and no movement.
0x07	1	Range	Section 3.8.11, “Range register"”	0x40	RW - Defines the absolute range within the position.

3.8.4 Control ID register

This read-only register contains the control identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-30. Control ID register

Encoding	Control Type
001	Keypad
010	Slider
011	Rotary
100	Analog slider
101	Analog rotary
110	Matrix
111	Reserved

NOTE

The control number is assigned in the system setup module. This value is unique for all controls, indicating that each control has a particular number regardless of its type. The first assigned control number is zero.

3.8.5 Control configuration

This register configures overall enablers of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R	Control Enabler	Callback Enabler	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
W								
Reset:	0	0	-	-	-	-	-	-

Figure 3-31. Control Configuration register

Table 3-30. Control configuration description

Signal	Description
Control Enabler	Global enabler for the control. This determines if the control electrodes are scanned for detection. The bit enables or disables all control electrodes in the Electrode enablers register. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function is called whenever an active event in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled

3.8.6 Dynamic Status register

This register describes the movement over the electrodes which are assigned to the controller.

Register Number = 0x02

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	Displacement					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-32. Dynamic Status register

Table 3-31. Dynamic Status register description

Signal	Description
Movement	Indicates whether the movement is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of the movement. This bit remains with its last value even if movement has stopped and is no longer detected. 1 — Incremental (from PositionX to PositionY, where X< Y) 0 — Decremental (from PositionX to PositionY, where X> Y)
Displacement	This value indicates the difference in positions from the last status to the new status. This indicates how many positions are advanced in the current movement direction. 0–63 — Number of positions.

3.8.7 Position register

The Position register contains an absolute position within a defined range.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R	Position							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-33. Static Status register

Table 3-32. Position register description

Signal	Description
Position	Indicates the calculated absolute position within a defined range. 0—*Range* - Number of the actual position

3.8.8 Events Control register

The Event Control register contains the bits used to enable or disable events that call the control callback function.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R W	Touch	Invalid Position	Reserved	Release Event Enabler	Hold Auto-Repeat Enabler	Hold Event Enabler	Movement Event Enabler	Initial Touch Event Enabler
Reset:	0	0	-	0	0	0	0	0

Figure 3-34. Events Control register

Table 3-33. Events Control register description

Signal	Description
Touch	This bit indicates whether the touch is detected on the control. 1 — Touch detected 0 — Touch not detected
Invalid Position	Indicates whether an invalid combination of touched electrodes is being detected. 1 — Invalid position detected 0 — Invalid position not detected
Release Event Enabler	If this bit is set and the callback function is enabled, the callback is called when all touched electrodes in the control are released. 1 — Release event enabled 0 — Release event disabled

Table 3-33. Events Control register description

Signal	Description
Hold Auto-repeat Enabler	If this bit is set, the hold event enabler and the callback function are enabled and the callback is called at the rate specified in the Auto-repeat Rate register as long as one valid position is detected as touched in the control and no movement is detected. 1 — Auto-repeat feature enabled 0 — Auto-repeat disabled
Hold Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function when the movement stops and, after a certain period of time, a constant touched position is detected. This time is configurable in the Movement Timeout Register. 1 — Event enabled 0 — Event disabled
Movement Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function each time a displacement is detected. 1 — Event enabled 0 — Event disabled
Initial Touch Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function whenever the control transitions from an idle to an active state. 1 — Event enabled 0 — Event disabled

NOTE

If the events are not enabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.8.9 Auto-repeat Rate register

The Auto-repeat Rate register contains the rate value whereby a hold event is reported when kept pressed without movement.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R W	Auto-repeat start							
Reset:	0	0	0	0	0	0	0	0

Figure 3-35. Auto-repeat Rate register

Table 3-34. Auto-repeat Rate register description

Signal	Description
Auto-repeat Rate	The decoder calls the control callback function at the specified rate as long as a valid position is continuously detected, the auto-repeat feature is enabled, and no displacement occurs. If this register is set to zero (0), the auto-repeat feature will be disabled in the events register. 0 — Auto-repeat feature disabled 1–254 — Callback is called every *n* task executions

NOTE

The Auto-repeat function works only if a hold event is presented. The hold event is delayed by the Movement Timeout function. If the Movement Timeout register is set to zero (0), the Auto-repeat function stops to generate a callback, because the hold event is no longer reported.

3.8.10 Movement Timeout register

The register value defines how long the movement event stays reported after a hold event is detected. The delay is defined in number of `TSS_Task()` executions. The hold event indicates an instance when the movement stops and a constant touched position is detected.

Register Number = 0x06

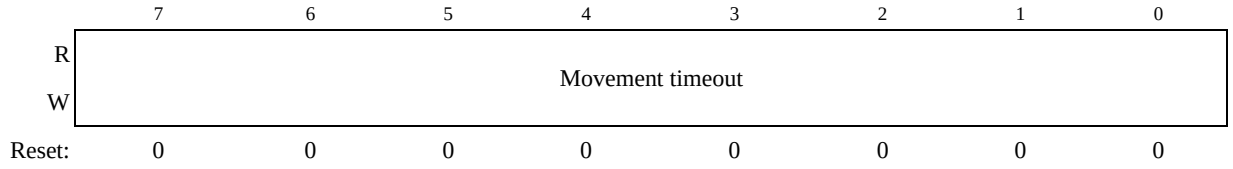

Figure 3-36. Movement Timeout register

Table 3-35. Movement Timeout register description

Signal	Description
Movement Timeout	Determines how many execution times the decoder must detect a no-displacement before reporting no movement and a hold event. 0 — No limit established. Movement is reported until the control is detected to be idle. 1–254 — Reports a hold event after n task executions.

3.8.11 Range register

The range register value defines the absolute range within the position.

Register Number = 0x06

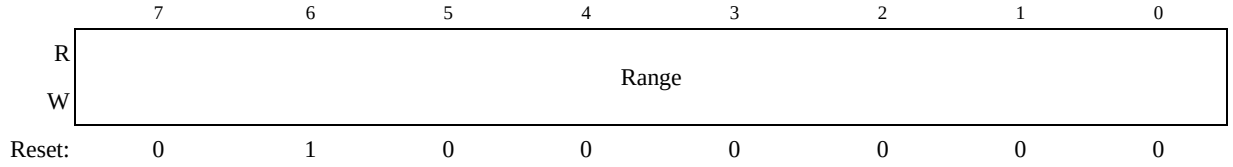

Figure 3-37. Range register

Table 3-36. Range register description

Signal	Description
Range	Defines the absolute range within the position. 2— 255 for Analog slider 3— 255 for Analog rotary

3.8.12 Callback function

The decoder module calls this function if an event occurs and if the user enables the callback function. Callback functions are assigned to controls in the system setup module. One callback may be assigned to several different controls in the system.

Function prototype

```
void CallbackFuncName(uint8_t u8ControlId)
```

The CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK fCallBack4
```

Where:

- *n* is the number of the control
- *fCallBack4* is a name example

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8ControlId	Any valid control ID of any control in the system	It indicates the control that generates the event. This parameter matches the controlled field in the control C&S register.

Return value

None

3.9 Matrix decoder API

This section describes Matrix Decoder API, Matrix C&S registers, the TSS_SetMatrixConfig() function used to write to these registers, and the TSS_GetMatrixConfig() function and data structure used for reading the register. This section also describes the callback function and its parameters for a Matrix control. Each Application Matrix control has corresponding Configuration and Status registers and a callback function.

3.9.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, call the TSS_SetSystemConfig function. This function is declared in the TSS_API.h file.

Function prototype

```
uint8_t TSS_SetMatrixConfig (TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter, uint8_t
u8Value)
```

Input parameters

Type	Name	Valid range and values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured which is stored in the first element of the control's C&S structure.
uint8_t	u8Parameter	Any of the parameter codes provided by matrix decoder	Code indicating the parameter configured
uint8_t	u8Value	Depends on the specific parameter configured	New desired value for the respective configuration registers

Return value

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration is executed successfully.
TSS_ERROR_MATRIX_ILLEGAL_PARAMETER	Configuration is not executed because of the illegal parameter number.
TSS_ERROR_MATRIX_ILLEGAL_VALUE	Configuration is not executed because of the illegal value.
TSS_ERROR_MATRIX_READ_ONLY_PARAMETER	Configuration is not executed whenever the user attempts to modify a read-only parameter.
TSS_ERROR_MATRIX_ILLEGAL_VALUE	Configuration is not executed becaus of the illegal value number.
TSS_ERROR_MATRIX_OUT_OF_RANGE	Configuration is not executed because the new value is out of range of the established boundaries
TSS_ERROR_MATRIX_ILLEGAL_CONTROL_TYPE	Configuration is not executed because the u8ControlId parameter does not match the control type structure the user is trying to modify.

3.9.2 Reading the Configuration and Status registers

The two options to access the Configuration and Status register structure involve using the TSS function declared in the TSS_API.h file and directly accessing the configuration and status structure for specific data. The following content describes the first option.

Function prototype

```
uint8_t TSS_GetMatrixConfig(TSS_CONTROL_ID u8ControlId, uint8_t u8Parameter);
```

Input parameters

Return value

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control's C&S structure
uint8_t	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds either to a control register unsigned byte value specified by the u8Parameter, or to error values defined in the file TSS_StatusCodes.h:

Return Value	Description
Any valid unsigned byte value	Value corresponds to a register unsigned byte value specified by the u8Parameter.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Read is not successful because of the illegal parameter.
TSS_ERROR_GETCONF_ILLEGAL_CONTROL_TYPE	Read is not successful because of the illegal control type.

The second option to access the Configuration and Status register structure involves direct access to the configuration and status structure for specific data. The application defines the structure name in the TSS_SystemSetup.h file by using the following definition:

```
#define TSS_Cn_STRUCTURE      cMatrix
```

Where *n* is the control number starting from zero (0) going up to the number of controls minus one.

The *cMatrix* name is an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_MATRIX_CONTROL ControlConfig;                  //Note 2
        const TSS_MATRIX_EVENTS Events;                          //Note 3
        const uint8_t AutoRepeatRate;
        const uint8_t MovementTimeout;
        const TSS_MATRIX_DYN DynamicStatusX;                     //Note 4
        const TSS_MATRIX_DYN DynamicStatusY;
        const uint8_t PositionX;
        const uint8_t PositionY;
        const uint8_t GestureDistanceX;
        const uint8_t GestureDistanceY;
        const uint8_t RangeX;
        const uint8_t RangeY;
  } TSS_CSMatrix
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.9.4, "Control ID register""

2. The TSS_MATRIX_CONTROL type is an eight bit-field structure described in Section 3.9.5, “Control configuration"”
3. The TSS_MATRIX_EVENTS type is an eight bit-field structure described in Section 3.9.6, “Events Control register"”
4. The TSS_MATRIX_DYN type is an eight bit-field structure described in Section 3.9.9, “Dynamic Status X register"”

Using the example of the structure named *cMatrix*, to read Control ID use:

```
Temp = cMatrix.ControlId;
```

3.9.3 Configuration and Status registers list

The following table describes the matrix control configuration and status registers.

Table 3-37. Matrix Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.9.4, “Control ID register"”	Application dependable	R — Displays the control type and control number.
0x01	1	ControlConfig	Section 3.9.5, “Control configuration"”	0x00	RW — This register configures overall enablers of the object.
0x02	1	Events	Section 3.9.6, “Events Control register"”	0x00	RW — Configures the events that call the callback function.
0x03	1	AutoRepeatRate	Section 3.9.7, “Auto-repeat Rate register"”	0x00	RW — Configures the rate at which keys are reported when they are kept pressed and no movement is detected.
0x04	1	MovementTimeout	Section 3.9.8, “Movement Timeout register"”	0x00	RW — Number of times the decoder must detect a no-displacement before reporting a hold event and no movement.
0x05	1	DynamicStatusX	Section 3.9.9, “Dynamic Status X register"”	0x00	R — Displays movement, direction, and displacement information on the X axis
0x06	1	DynamicStatusY	Section 3.9.10, “Dynamic Status Y register"”	0x00	R — Displays movement, direction, and displacement information on the Y axis
0x07	1	PositionX	Section 3.9.11, “Position X register"”	0x00	R — Displays absolute position information within a defined range on the X axis.

Table 3-37. Matrix Control Configuration and Status registers (continued)

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x08	1	PositionY	Section 3.9.12, “Position Y register"”	0x00	R — Displays absolute position information within a defined range on the Y axis.
0x09	1	GestureDistanceX	Section 3.9.13, “Gesture distance X register"”	0x00	R — Displays relative distance between two or more positions within a defined range on the X axis.
0x0A	1	GestureDistanceY	Section 3.9.14, “Gesture distance Y register"”	0x00	R — Displays relative distance between two or more positions within a defined range on the Y axis.
0x0B	1	RangeX	Section 3.9.15, “Range X register"”	0x40	R — Defines the absolute range on the X axis.
0x0C	1	RangeY	Section 3.9.15, “Range X register"”	0x40	R — Defines the absolute range on the Y axis

3.9.4 Control ID register

This read-only register contains the control identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-38. Control ID register

Encoding	Control Type
001	Keypad
010	Slider
011	Rotary
100	Analog slider
101	Analog rotary

Encoding	Control Type
100	Matrix
111	Reserved

NOTE

The control number is assigned in the system setup module. This value is unique for all controls, indicating that each control has a particular number regardless of its type. The first assigned control number is zero.

3.9.5 Control configuration

This register configures overall enablers of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R W	Control Enabler	Callback Enabler	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Reset:	0	0	0	0	0	0	0	0

Figure 3-39. Control Configuration register

Table 3-38. Control configuration description

Signal	Description
Control Enabler	Global control enabler. This determines whether the control electrodes are scanned for detection. The bit enables or disables all control electrodes in the Electrode enablers register. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function is called when one of the active events in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled

3.9.6 Events Control register

The Event Control register contains the bits used to enable or disable events that call the control callback function.

Register Number = 0x02

	7	6	5	4	3	2	1	0
R W	Touch	Gesture	Gestures Enabler	Release Event Enabler	Hold Auto-Repeat Enabler	Hold Event Enabler	Movement Event Enabler	Initial Touch Event Enabler
Reset:	0	0	0	0	0	0	0	0

Figure 3-40. Events Control register

Table 3-39. Events Control register description

Signal	Description
Touch	This bit indicates whether the touch has been detected on the control. 1 — Touch detected 0 — Touch not detected
Gesture	This bit indicates whether the gesture has been detected on the control. The gesture is reported if at least two isolated touches are detected within the control. 1 — Gesture detected 0 — Gesture not detected
Gestures Enabler	If this bit is set, the gesture evaluation is enabled and the information about gesture starts writing into the Gesture Distance registers. This option also enables generating callbacks on all enabled events. However, the event source is Gesture Distance register and not the Position register. 1 — Gestures enabled 0 — Gestures disabled
Release Event Enabler	If this bit is set and the callback function is enabled, the callback is called when all touched electrodes in the control are released. 1 — Release event enabled 0 — Release event disabled
Hold Auto-repeat Enabler	If this bit is set, the hold event enabler and the callback function are enabled, the callback is called at the rate specified in the Auto-repeat Rate register as long as one valid position is detected as touched in the control and no movement is detected. 1 — Auto-repeat feature enabled 0 — Auto-repeat disabled
Hold Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function when the movement stops and a constant touched position is detected after a certain period of time. This time is configurable in the Movement Timeout Register. 1 — Event enabled 0 — Event disabled
Movement Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function every time a displacement is detected. 1 — Event enabled 0 — Event disabled
Initial Touch Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function when the control transitions from an idle to an active state. 1 — Event enabled 0 — Event disabled

NOTE

If no events are enabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.9.7 Auto-repeat Rate register

The Auto-repeat Rate register contains the rate value where a hold event is reported when kept pressed without movement.

Register Number = 0x03

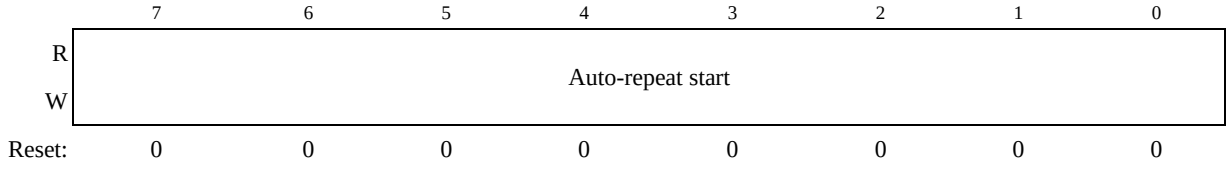

Figure 3-41. Auto-repeat Rate register

Table 3-40. Auto-repeat Rate register description

Signal	Description
Auto-repeat Rate	The decoder calls the control callback function at the specified rate as long as a valid position is continuously detected, the auto-repeat feature is enabled and no displacement occurs. If this register is set to zero (0), the auto-repeat feature is disabled in the events register. 0 — Auto-repeat feature disabled 1–254 — Callback is called every *n* task executions

NOTE

The Auto-repeat function works only if a hold event is presented. The hold event is delayed by the Movement Timeout function. If the Movement Timeout register is set to zero (0), the Auto-repeat function stops to generate a callback, because the Hold event is no longer reported.

3.9.8 Movement Timeout register

The register value defines how long the movement event stays reported after a hold event is detected. The delay is defined in a number of `TSS_Task()` executions. The hold event represents an instance when the movement stops and a constant touched position is detected.

Register Number = 0x04

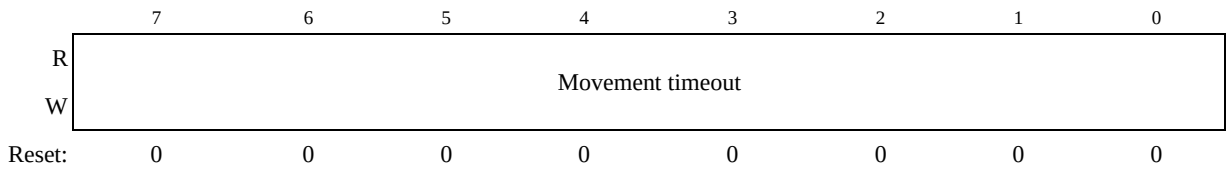

Figure 3-42. Movement Timeout register

Table 3-41. Movement Timeout register description

Signal	Description
Movement Timeout	Determines how many execution times the decoder must detect a no-displacement before reporting no movement and reporting a hold event. 0 — No limit established. Movement is reported until the control is detected as idle. 1–254 — Reports a hold event after n task executions.

3.9.9 Dynamic Status X register

This register describes the movement on the horizontal X axis over the electrodes assigned to the control. The information is based on a Position X or a Gesture Distance X register information.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	Displacement					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-43. Dynamic Status X register

Table 3-42. Dynamic Status X register description

Signal	Description
Movement	Indicates if any movement on the X axis is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of the movement on X axis. This bit remains with its last value even if movement has stopped and is no longer detected. 1 — Incremental (from PositionN to PositionM, where N < M) 0 — Decremental (from PositionN to PisitionM, where N > M)
Displacement	This value indicates the difference in positions on X axis from the last status to the new status. This indicates how many positions have been advanced in the current movement direction. 0–63 — Number of positions.

3.9.10 Dynamic Status Y register

This register describes the movement on Y vertical axis over the electrodes assigned to the control. The information is based on a position Y or a gesture distance Y register information.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	Displacement					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-44. Dynamic Status Y register

Table 3-43. Dynamic Status Y register description

Signal	Description
Movement	Indicates whether any movement on Y axis is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the movement direction on Y axis. This bit remains with its last value even if movement has stopped and is no longer detected. 1 — Incremental (from PositionN to PositionM, where N < M) 0 — Decremental (from PositionN to PisitionM, where N > M)
Displacement	This value indicates the difference in positions on Y axis from the last status to the new status. This indicates how many positions have been advanced in the current movement direction. 0–63 — Number of positions.

3.9.11 Position X register

This Position X register contains an absolute analog position within a defined range on X horizontal axis.

Register Number = 0x07

	7	6	5	4	3	2	1	0
R	Position X							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-45. Position X register

Table 3-44. Position X register description

Signal	Description
Position	Indicates the calculated absolute analog position within a defined range. 0—*Range* - Number of the actual position

3.9.12 Position Y register

The Position Y register contains an absolute analog position within a defined range on Y vertical axis.

Register Number = 0x08

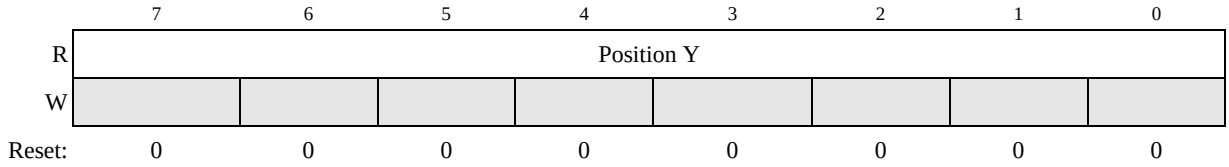

Figure 3-46. Position Y register

Table 3-45. Position Y register description

Signal	Description
Position	Indicates the calculated absolute analog position within a defined range. 0—*Range* - Number of the actual position

3.9.13 Gesture distance X register

The Gesture Distance X register contains a calculated maximum distance between gesture analog positions on X horizontal axis. The gesture event is reported if at least two isolated touches are detected within a control range.

Register Number = 0x09

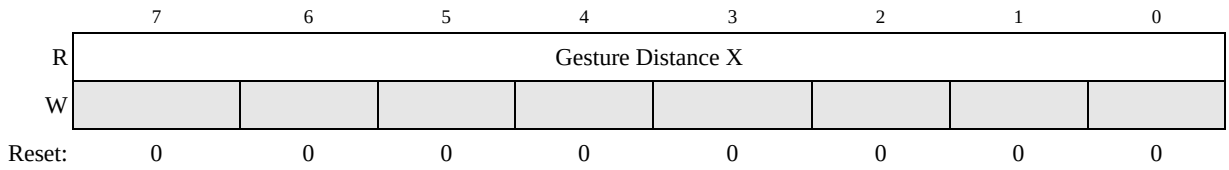

Figure 3-47. Gesture Distance X register

Table 3-46. Gesture Distance X register description

Signal	Description
Position	Indicates a calculated maximum distance between gesture analog positions on X horizontal axis. 0—*Range* - Number of the actual position

3.9.14 Gesture distance Y register

The Gesture Distance Y register contains a calculated maximum distance between gesture analog positions on Y vertical axis. The gesture event is reported if at least two isolated touches are detected within a control range.

Register Number = 0x09

	7	6	5	4	3	2	1	0
R	Gesture Distance Y							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-48. Gesture Distance Y register

Table 3-47. Gesture Distance Y register description

Signal	Description
Position	Indicates calculated maximum distance between gesture analog positions on Y vertical axis. 0—*Range* - Number of the actual position

3.9.15 Range X register

The range register value defines the absolute range within the position on X horizontal axis.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R	Range X							
W								
Reset:	0	1	0	0	0	0	0	0

Figure 3-49. Range X register

Table 3-48. Range X register description

Signal	Description
Range	Defines the absolute range within the position on X horizontal axis. 2— 255 - Range value

3.9.16 Range Y register

The range register value defines the absolute range within the position on Y vertical axis.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R	Range Y							
W								
Reset:	0	1	0	0	0	0	0	0

Figure 3-50. Range Y register

Table 3-49. Range Y register description

Signal	Description
Range	Defines the absolute range within the position on Y vertical axis. 2— 255 - Range value

3.9.17 Matrix callback function

The decoder module calls this function if an event occurs and the user enables the callback function. Callback functions are assigned to controls in the system setup module. One callback may be assigned to several different controls in the system.

Function prototype

```
void CallbackFuncName(uint8_t u8ControlId)
```

The CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK fCallBack5
```

Where:

- *n* is the number of the control
- *fCallBack5* is a name example

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8ControlId	Any valid control ID of any control in the system	It indicates the control that generates the event. This parameter matches the controlled field in the control C&S register.

Return value

None

Chapter 4
Library Intermediate Layer Interfaces

The TSS library architecture supports module implementation depending on the application requirements. Applications can be customized by using a suitable library layer. To implement modules, it is necessary to understand how modules exchange information and communicate with each other. This section describes the parameters and functions used by the library to establish communication between layers. For more details about the TSS library architecture, see Section 1.3, “Library architecture.”

4.1 Capacitive sensing and key detector interface

The capacitive sensing layer senses each electrode declared in the system. Based on the information provided by the capacitive sensing layer, the key detector layer determines whether an electrode is being pressed or not. Table 4-1 shows the global value where the charging time is stored.

Table 4-1. Global variable description

Parameter	Description
tss_u16CapSample	Global 16 bits variable used to store the electrode charging time.

The tss_u16CapSample global variable passes the acquired sensing data from the capacitive sensing layer to the key detector layer. When the capacitive sensing layer senses an electrode, it stores the charging time information in the tss_u16CapSample global variable. To use a custom capacitive sensing module, the acquired value from the sensing module is passed to the upper layer by storing this value in the tss_u16CapSample global variable.

4.1.1 Electrode sampling

The exchange of information between the capacitive sensing and key detector layers is accomplished by using the sensing function.

Function prototype

```
uint8_t (* const tss_faSampleElectrode[])(uint8_t u8ElecNum, uint8_t u8Command)
```

Function description

The key detector layer calls the sensing function each time it needs to sense an electrode. The function

```
uint8_t (* const tss_faSampleElectrode[])(uint8_t u8ElecNum, uint8_t u8Command)
```

is used to select the appropriate sensing function according to the defined low level method (TSI, GPIO, etc.) The sensing function allows both layers to exchange parameters. When the key detector layer calls this function, it provides the electrode number that needs to be scanned and a command for a measurement routine. When the sensing is completed, the capacitive sensing layer returns the sensing state to the upper layer.

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8ElecNum	Any valid Electrode number	Number of electrodes to be scanned by the capacitive sensing layer
uint8_t	u8Command	Any valid command: TSS_SAMPLE_COMMAND_DUMMY TSS_SAMPLE_COMMAND_RESTART TSS_SAMPLE_COMMAND_PROCESS TSS_SAMPLE_COMMAND_RECALIB TSS_SAMPLE_COMMAND_GET_NEXT_ELECTRODE TSS_SAMPLE_COMMAND_ENABLE_ELECTRODE TSS_SAMPLE_COMMAND_RESTART_NOISE TSS_SAMPLE_COMMAND_GET_NOISE_VALUE TSS_SAMPLE_COMMAND_SET_LOWLEVEL_CONFIG TSS_SAMPLE_COMMAND_GET_LOWLEVEL_CONFIG	These commands are used for management of the measurement process and setting of the electrode from the upper level.

Return value

The function returns the resulting status of the electrode sensing to the key detector layer.

Possible return values are described below:

Return Value	Description
TSS_SAMPLE_STATUS_OK	Electrode sensing is successfully executed.
TSS_SAMPLE_STATUS_PROCESSING	The electrode sensing process is still running.
TSS_SAMPLE_STATUS_CALIBRATION_CHANGED	Electrode is recalibrated
TSS_SAMPLE_RECALIB_REQUEST_LOCAP	Electrode sensing is not successful becacuse of the small capacitance. Hardware recalibration is needed. If the TSI method is used, the measured signal is below 20% of the required TSI resolution.
TSS_SAMPLE_RECALIB_REQUEST_HICAP	Electrode sensing is not successful done because of the high capacitance. Hardware recalibration is needed. If the TSI method is used, the measured signal is higher than 0xFFFF minus the 20% of required TSI resolution.
TSS_SAMPLE_ERROR_SMALL_TRIGGER_PERIOD	A small trigger period is detected.
TSS_SAMPLE_ERROR_CHARGE_TIMEOUT	Electrode sensing is not successful because of the charge timeout.
TSS_SAMPLE_ERROR_SMALL_CAP	Electrode sensing is not successful because of the small capacity.
TSS_SAMPLE_ERROR_RESULT_NA	A requested value, or a feature is not available.

When an error occurs, the key detector layer determines whether the sense capacitance was too small or too large, whether the hardware recalibration is needed, or whether another, non-standard, state is detected.

4.1.2 Low-level initialization

The initialization of low level hardware from the Key Detector layers is performed by the Sensor Init function.

Function prototype

```
uint8_t TSS_SensorInit(uint8_t u8Command)
```

Function description

The key detector layer calls the Sensor Init function each time it needs to change either the configuration of the low level hardware or an electrode. The function `uint8_t TSS_SensorInit(uint8_t u8Command)` is used to reconfigure the entire low level, all electrodes, regardless of the measurement method. The Sensor Init function allows both layers to exchange parameters. When the key detector layer calls this function, it provides a command for this function. When the configuration of low level is completed, the Sensor Init function returns the state to the upper layer.

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8Command	Any valid command: • TSS_INIT_COMMAND_DUMMY • TSS_INIT_COMMAND_INIT_MODULES • TSS_INIT_COMMAND_ENABLE_ELECTRODES • TSS_INIT_COMMAND_SET_NSAMPLES • TSS_INIT_COMMAND_INIT_TRIGGER • TSS_INIT_COMMAND_SW_TRIGGER • TSS_INIT_COMMAND_INIT_LOWPOWER • TSS_INIT_COMMAND_GOTO_LOWPOWER • TSS_INIT_COMMAND_RECALIBRATE • TSS_INIT_COMMAND_ENABLE_LOWPOWER_CALIB • TSS_INIT_COMMAND_MODE_CAP • TSS_INIT_COMMAND_MODE_NOISE • TSS_INIT_COMMAND_MODE_GET	These commands are used to set the entire low level hardware from the upper level.

Return value

The function returns the resulting status of the configuration to the key detector layer.

Possible return values are described below.

Return Value	Description
TSS_INIT_STATUS_OK	Configuration is successful.
TSS_INIT_STATUS_LOWPOWER_SET	Configuration of Low Power function is successful.
TSS_INIT_STATUS_LOWPOWER_ELEC_SET	Configuration of Low Power electrode is successful.
TSS_INIT_STATUS_TRIGGER_SET	Configuration of Trigger is successful.
TSS_INIT_STATUS_AUTOTRIGGER_SET	Configuration of Auto Trigger mode is successful.
TSS_INIT_STATUS_CALIBRATION_CHANGED	Calibration is changed.
TSS_INIT_ERROR_RECALIB_FAULT	Calibration is unsuccessful.

4.2 Decoder interface

The decoders provide the highest level of abstraction in the library. In this layer, the key detector information about touched and untouched electrodes is interpreted to present the control status in a behavioral way. Decoder-related code exists only once in memory. Decoders are similar to classes of an object oriented language. Because each control has a decoder associated with it, the control becomes an instance of the decoder (an object). However, not all decoders are necessarily instantiated in every system. The TSS library supports Rotary, Slider, Analog rotary, Analog slider, Matrix, and Keypad inside the precompiled library files. Additionally, there is an interface to support other decoders externally. For more information about the decoder functionality, see the *Touch Sensing User Guide*.

4.2.1 Decoder main function

The exchange of information between the decoder and key detector layer is performed by using the following function.

Function prototype

```
uint8_t (* const tss_faDecoders[])(uint8_t u8CtrlNum, uint16_t u16TouchFlag, uint8_t
u8NumberOfElec, uint8_t u8Command)
```

Function description

The key detector layer calls the decoder function everytime electrodes are measured. The decoder function allows both layers to exchange parameters. When the key detector layer calls this function, it provides the control number which is processed, pointer to the data which provides the electrode state information with signal change, and a command. When the decoder function finishes processing, it returns the status to the lower layer.

The electrode state buffer always provides a pointer to the data of the electrodes related to the control. The data consists of two 16-bit variables. The first variable provides information about the touch state of the electrodes in the form of bit flags relative to the control (logic one is touch state, zero is release state). The second variable provides information about the signal change of the electrodes in the form of bit flags relative to the control (logic one represents signal change).

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8CtrlNum	Any valid control number	Number of control to be processed
uint16_t	u16TouchFlag	16 bit mask	Unsigned integer variable where each bit represents touch status of electrodes assigned to the control
uint8_t	u8NumberOfElec	1 - 16	Number of electrodes assigned to the control
uint8_t	u8Command	Any valid command: TSS_DECODER_COMMAND_DUMMY TSS_DECODER_COMMAND_PROCESS TSS_DECODER_COMMAND_INIT	These commands are used for decoder process management and decoder setting from lower level

Return value

The function returns the resulting status of the decoder to the key detector layer.

Possible return values are described below:

Return Value	Description
TSS_DECODER_STATUS_OK	Decoder processing successfully finished
TSS_DECODER_ERROR_ILLEGAL_CONTROL_TYPE	Called decoder type does not match with control type
TSS_DECODER_STATUS_BUSY	Decoder is not ready to perform processing.

4.2.2 Writing the decoder schedule counter

The Key Detector provides an internal counter that can schedule decoder calls periodically after a defined number of TSS_Task measurement cycles. The initialization of the control schedule counter is performed by this function.

Function prototype

```
void TSS_KeyDetectorSetControlScheduleCounter(uint8_t u8CntrlNum, uint8_t u8Value)
```

Function description

The key detector layer decrements the control schedule counter after each electrode measurement cycle is finished.

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8CntrlNum	Any valid control number	Control index
uint8_t	u8Value	1-255	Number of TSS_Task measurement cycles

Return value

None

4.2.3 Reading the decoder schedule counter

The key detector provides an internal counter that can schedule a decoder call after a defined number of TSS_Task measurement cycles. The status of this counter can be read by this function.

Function prototype

```
uint8_t TSS_KeyDetectorGetControlScheduleCounter(uint8_t u8CntrlNum)
```

Function description

The key detector layer decrements control schedule counter after each measurement cycle. The function returns actual state of the counter

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8CntrlNum	Any valid control number	Control index

Return value

The status of the control schedule counter.

4.2.4 Reading the instant delta in the control

The key detector provides information about the electrode instant delta value.

Function prototype

```
int8_t TSS_KeyDetectorGetInstantDelta(uint8_t u8ElecNum)
```

Function description

The function returns instant delta value for a defined electrode.

Input parameters

Type	Name	Valid range/values	Description
uint8_t	u8ElecNum	Any valid electrode number	Number of instant delta electrode

Return value

The instant delta value of a defined electrode.

Chapter 5
C++ wrapper

This chapter describes the C++ wrapper of the TSS library. It extends the C library with simplified API. This API makes usaging the TSS more convenient.

All definitions are contained within namespace called “tss”. Doxygen documentation is part of all headers. The definition of available flags, constants, and commands are in TSS_cpp_const header file.

The following are the application requirements:

- If any control is used, TSS_USE_CONTROL_PRIVATE_DATA should be enabled. It’s used to store the C++ “this” pointer for a control.
- Faults are managed by callbackFaultHelper function which should be set as onFault callback
- Each control should register callbackControlHelper function as general control callback

5.1 TSSSystem class

Class TSSSystem represents the main TSS class used for system configuration. Because this class is a singleton, only one instance of the TSSSystem exists in an application.

This class provides the API for configuring the TSS. A TSSSystem object can be used without any controls (polling electrode status method).

5.1.1 TSSTask

The TSSTask method should be periodically called to provide a CPU time to the TSS.

Function prototype

```
static uint8_t TSSTask(void)
```

Function description

Static method which invokes the library TSS_Task() function.

Input parameters

None

Return value

The return value is uint8_t with the following possible return values:

Return Value	Description
TSS_STATUS_OK	TSS task finished the measurement of all electrodes and has evaluated the values
TSS_STATUS_PROCESSING	Measurement of electrodes in progress or the TSS task has not finished the evaluation

5.1.2 isElecTouched

Returns boolean value if the specified electrode is touched.

Function prototype

```
bool isElecTouched(uint8_t elec_num) const
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode to be checked

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	An electrode is touched
false	An electrode is released

5.1.3 isElecEnabled

Returns boolean value if the specified electrode is enabled.

Function prototype

```
bool isElecEnabled(uint8_t elec_num) const
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode to be checked

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	An electrode is enabled
false	An electrode is disabled

5.1.4 enableElec

Enables the specified electrode.

Function prototype

```
bool enableElec(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The electrode is enabled.
false	Failed

5.1.5 disableElec

Disables the specified electrode.

Function prototype

```
bool disableElec(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The electrode is disabled.
false	Failed

5.1.6 setSensitivity

Sets a sensitivity value for a specified electrode.

Function prototype

```
bool setSensitivity(uint8_t elec_num, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode
uint8_t	value	1-127	The sensitivity value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The sensitivity is set.
false	Failed

5.1.7 getSensitivity

Retrieves a sensitivity value for a specified electrode.

Function prototype

```
uint8_t getSensitivity(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of an electrode.

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
0	Non-valid value
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Bad parameter
A value between 2-127	Returned sensitivity value

5.1.8 enableDCTracker

Enables DC tracker for the specified electrode.

Function prototype

```
bool enableDCTracker(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The DC tracker is enabled for a specified electrode
false	Failed

5.1.9 disableDCTracker

Disables DC tracker for a specified electrode.

Function prototype

```
bool disableDCTracker(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The DC tracker is disabled for the specified electrode.
false	Failed

5.1.10 setBaseline

Sets a baseline for the specified electrode.

Function prototype

```
bool setBaseline(uint8_t elec_num, uint16_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode
uint16_t	value	0 - UINT16 MAX	The baseline value

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The baseline is set.
false	Failed

5.1.11 getBaseline

Returns a value of the baseline for a specified electrode.

Function prototype

```
uint16_t getBaseline(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The number of the electrode

Return value

The return value is boolean with the following possible return values:

Return Value	Description
1 - UINT16 MAX	The baseline value
0	Failed

5.1.12 enable

Enables TSS System features.

Function prototype

```
bool enable(TSSSystemFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSSystemFlag`	`flag`	TSSSystemFlag enum values	The TSS System feature to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The feature is set.
false	Failed

5.1.13 disable

Disables TSS System features.

Function prototype

```
bool disable(TSSSystemFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSSystemFlag`	`flag`	TSSSystemFlag enum values	The TSS System feature to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The feature is set.
false	Failed

5.1.14 set

Sets a specified value in the TSS System register.

Function prototype

```
bool set(TSSSystemConfig command, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSSystemConfig`	`command`	`TSSSystemConfig enum value`	The TSS System register to be set
uint8_t	`value`	Valid range for the specific register	The value to be stored

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The register value is updated.
false	Failed

5.1.15 get

Retrieves a specified value from the TSS System register.

Function prototype

```
uint16_t get(TSSSystemConfig command)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSSystemConfig	command	TSSSystemConfig enum value	The TSS System register to be retrieved

Return value

The return value corresponds to an actual uint16_t value of a register specified by the first parameter - the command.

5.1.16 callbackSysFaults

This is a callback which invokes the user's fault callback and the onFault callback if an error occurs.

Function prototype

```
void callbackSysFaults(TSSSystemEvent event, uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSSystemEvent	event	TSSSystemEvent enum value	A specific error which has occurred.
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The electrode number

Return value

None

5.1.17 onFault

This is a virtual callback which invokes the user's callback and the onFault callback if an error occurs.

Function prototype

```
virtual void onFault(TSSSystemEvent event, uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSSystemEvent	event	TSSSystemEvent enum value	A specific fault which has occurred
uint8_t	elec_num	0 - TSS_N_ELECTRODES	The electrode number

Return value

None

5.1.18 regCallback

This method registers user callback for a specified event.

Function prototype

```
bool regCallback(TSS_fSystemCallback callback, TSSSystemEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSS_fSystemCallback`	`callback`	`Pointer to a function`	A function invoked when a specified state occurs
`TSSSystemEvent`	`state`	One of values from `TSSSystemEvent`	The TSS System's event to be registered

Return value

None

5.1.19 unregCallback

This method unregisters user callback for a specified event.

Function prototype

```
bool unregCallback(TSSSystemEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSSystemEvent`	`state`	One of values from `TSSSystemEvent`	The TSS System event to be unregistered

Return value

None

5.2 TSSControlFactory class

Class TSSControlFactory is a factory interface for handling TSS controls.

5.2.1 createTSSControl

Creates new class which represents the TSS control or returns pointer to an already created class.

Function prototype

```
static TSSControl* createTSSControl(uint8_t control_number = 0)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	control_number	0-15	The control number which corresponds to a control in the TSSSystemSetup header file

Return value

A pointer to the newly created class which represents the TSS control.

5.2.2 getTSSControl

This method returns a pointer to the existing class representing the TSS control.

Function prototype

```
static TSSControl* getTSSControl(uint8_t control_number)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	control_number	0-15	The control number which corresponds to a control in the TSSSystemSetup header file

Return value

A pointer to the existing class representing the TSS control.

5.3 TSSControl class

TSSControl class defines an interface for TSS controls. This is a base class for specific TSS control classes. It does not provide any registration of callbacks because they are specific for each type of control. See a specific control class for implementation.

5.3.1 callbackControl

An interface for a specific callback handler for a control.

Function prototype

```
virtual void callbackControl(void) = 0
```

Input parameters

None

Return value

None

5.3.2 enable

An interface to enable specific control flags.

Function prototype

```
virtual bool enable(TSSControlFlag flag) = 0
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of values from TSSControlFlag	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated
false	Failed

5.3.3 disable

An interface to disable specific control flags.

Function prototype

```
virtual bool disable(TSSControlFlag flag) = 0
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of values from TSSControlFlag	TSSControlFlag to be disabled

Return value

The return value is a bool with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.3.4 set

An interface to set specific control flags.

Function prototype

```
virtual bool set(TSSControlConfig config, uint8_t value) = 0
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	config	One of values from TSSControlFlag	TSSControlFlag to be disabled
uint8_t	value	A valid value for specific control flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.3.5 get

An interface to get specific control flags.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const = 0
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	config	One of values from TSSControlFlag	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.4 TSSKeypad class

The class TSSKeypad defines the implementation of the keypad control.

5.4.1 callbackControl

This is the callback handler for the keypad control.

Function prototype

```
virtual void callbackControl(void)
```

Input parameters

None

Return value

None

5.4.2 enable

A method to enable the keypad control flag.

Function prototype

```
bool enable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of the values from the TSSControlFlag which is valid for the keypad control	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.4.3 disable

A method to disable the keypad control flag.

Function prototype

```
bool disable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of the values from the TSSControlFlag which is valid for the keypad control	TSSControlFlag to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.4.4 set

A method to set the keypad control values.

Function prototype

```
virtual bool set(TSSControlConfig command, uint8_t value) = 0
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	command	One of the values from the TSSControlFlag which is valid for the keypad control	TSSControlFlag to be disabled
uint8_t	value	A valid value for specific control flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The value is updated.
false	Failed

5.4.5 get

A method to retrieve the keypad control value.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlConfig	config	One of values from TSSControlFlag which is valid for the keypad control	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.4.6 getControlStruct

A method which returns a pointer to the keypad control structure.

Function prototype

```
const TSS_CSKeypad* getControlStruct(void) const
```

Input parameters

None

Return value

A pointer to the Keypad control structure.

5.4.7 regCallback

This method registers user callback for a specified event.

Function prototype

```
bool regCallback(fKeypadCallback callback, KeypadEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
fKeypad Callback	callback	Pointer to a function	A function to be invoked when the specified state occurs
Keypad Event	state	One of values from KeypadEvent	The keypad event to be registered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback is registered.
false	Failed

5.4.8 unregCallback

This method unregisters user callback for a specified event.

Function prototype

```
bool unregCallback(KeypadEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSSystemEvent	state	One of the values from the KeypadEvent	The keypad event to be unregistered

Return value

None

5.4.9 onTouch

A callback which is invoked when a touch flag is enabled and a touch event occurs .

Function prototype

```
virtual void onTouch(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	`elec_num`	Within the number of electrodes defined in System setup header file	The number of the electrode

Return value

None

5.4.10 onRelease

A callback which is invoked when the release flag is enabled and a release event occurrs .

Function prototype

```
virtual void onRelease(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	Within the number of electrodes defined in System setup header file	The number of the electrode

Return value

None

5.4.11 onExceededKeys

A callback which is invoked when an exceeded flag is enabled and maximum touch regsiter value is exceeded.

Function prototype

```
virtual void onExceededKeys(uint8_t max_touches)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	max_touches	0-255	The number of the maximum touches register

Return value

None

5.5 TSSASlider class

The class TSSASlider defines the implementation of the slider control.

5.5.1 callbackControl

This is the callback handler for the analog slider control.

Function prototype

```
virtual void callbackControl(void)
```

Input parameters

None

Return value

None

5.5.2 enable

A method to enable the analog slider control flag.

Function prototype

```
bool enable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of the values from the TSSControlFlag which is valid for the analog slider control	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.5.3 disable

A method to disable the analog slider control flag.

Function prototype

```
bool disable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSControlFlag`	`flag`	One of the values from the TSSControlFlag which is valid for the analog slider control	TSSControlFlag to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.5.4 set

A method to set the analog slider control values.

Function prototype

```
virtual bool set(TSSControlConfig command, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	command	One of the values from the TSSControlFlag which is valid for the analog slider control	TSSControlFlag to be disabled
uint8_t	value	A valid value for analog slider control flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The value is updated.
false	Failed

5.5.5 get

A method to retrieve the analog slider control value.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlConfig	config	One of the values from the TSSControlFlag which is valid for the slider control	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.5.6 getControlStruct

A method which returns a pointer to the control structure of the analog slider.

Function prototype

```
const TSS_CSASlider* getControlStruct(void) const
```

Input parameters

None

Return value

A pointer to the control structure of the analog slider.

5.5.7 regCallback

This method registers user callback for the specified event.

Function prototype

```
bool regCallback(fASliderCallback callback, ASliderEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
`fASliderCallback`	callback	A pointer to a function	A function to be invoked when the specified state occurs
`ASliderEvent`	state	One of values from ASliderEvent	An analog slider event to be registered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback is registered.
false	Failed

5.5.8 unregCallback

This method unregisters user callback for a specified event.

Function prototype

```
bool unregCallback(ASliderEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
ASlider Event	state	One of values from ASliderEvent	The analog slider event to be unregistered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback is unregistered.
false	Failed

5.5.9 onInitialTouch

A callback which is invoked when the initial touch flag is enabled and an initial touch event occurrs .

Function prototype

```
virtual void onInitialTouch(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	Within the number of electrodes defined in System setup header file	The number of the electrode

Return value

None

5.5.10 onAllRelease

A callback which is invoked when the release flag is enabled and all electrodes assigned to a control have entered a release state.

Function prototype

```
virtual void onAllRelease(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.5.11 onMovement

A callback which is invoked when the movement flag is enabled and a movement has been detected.

Function prototype

```
virtual void onMovement(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.6 TSSARotary class

The class TSSARotary defines the implementation of the analog rotary control.

5.6.1 callbackControl

This is the callback handler for the analog rotary control.

Function prototype

```
virtual void callbackControl(void)
```

Input parameters

None

Return value

None

5.6.2 enable

A method to enable the control flag of the analog rotary.

Function prototype

```
bool enable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of the values from the TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.6.3 disable

A method to disable the control flag of the analog rotary.

Function prototype

```
bool disable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of the values from the TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.6.4 set

A method to set the control value of the analog rotary.

Function prototype

```
virtual bool set(TSSControlConfig command, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	command	One of the values from the TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be disabled
uint8_t	value	A valid value for analog rotary control's flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The value is updated.
false	Failed

5.6.5 get

A method to retrieve the control value of the analog rotary.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlConfig	config	One of the values from the TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid.
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.6.6 getControlStruct

A method which returns a pointer to the control structure of the analog rotary.

Function prototype

```
const TSS_CSARotary* getControlStruct(void) const
```

Input parameters

None

Return value

A pointer to the control strucutre of the analog rotary.

5.6.7 regCallback

This method registers user callback for a specified event.

Function prototype

```
bool regCallback(fARotaryCallback callback, ARotaryEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
fARotaryCallback	callback	Pointer to a function	A function to be invoked when a specified state occurs
ARotaryEvent	state	One of values from ARotaryEvent	A rotary event to be registered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback is registered.
false	Failed

5.6.8 unregCallback

This method unregisters the user callback for a specified event.

Function prototype

```
bool unregCallback(ARotaryEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
ARotaryEvent	state	One of the values from the ARotaryEvent	The analog rotary event to be unregistered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback is unregistered.
false	Failed

5.6.9 onInitialTouch

A callback which is invoked when the initial touch flag is enabled and an initial touch event occurs .

Function prototype

```
virtual void onInitialTouch(uint8_t elec_num)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	elec_num	Within the number of electrodes defined in System setup header file	The number of the electrode

Return value

None

5.6.10 onAllRelease

A callback which is invoked when the release flag is enabled and all electrodes assigned to a control enter a release state.

Function prototype

```
virtual void onAllRelease(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.6.11 onMovement

A callback which is invoked when the movement flag is enabled and a movement is detected.

Function prototype

```
virtual void onMovement(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.7 TSSSlider class

The class TSSASlider defines the implementation of the slider control.

5.7.1 callbackControl

This is the callback handler for the slider control.

Function prototype

```
virtual void callbackControl(void)
```

Input parameters

None

Return value

None

5.7.2 enable

A method to enable the control flag of the slider.

Function prototype

```
bool enable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of the values from the TSSControlFlag which is valid for the slider control	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.7.3 disable

A method to disable the slider control flag.

Function prototype

```
bool disable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSControlFlag`	`flag`	One of the values from the TSSControlFlag which is valid for the slider control	TSSControlFlag to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag is updated.
false	Failed

5.7.4 set

A method to set the slider control values.

Function prototype

```
virtual bool set(TSSControlConfig command, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	command	One of the values from the TSSControlFlag which is valid for the slider control	TSSControlFlag to be disabled
uint8_t	value	A valid value for slider control's flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The value was updated
false	Failed

5.7.5 get

A method to retrieve the slider's control value.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlConfig	config	One of values from TSSControlFlag which is valid for the slider control	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.7.6 getControlStruct

A method which returns a pointer to the slider's control structure.

Function prototype

```
const TSS_CSSlider* getControlStruct(void) const
```

Input parameters

None

Return value

A pointer to the slider's control structure.

5.7.7 regCallback

This method registers user's callback for a specified event.

Function prototype

```
bool regCallback(fSliderCallback callback, SliderEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
`fSliderCallback`	callback	A pointer to a function	A function to be invoked when the specified state occurs
`SliderEvent`	state	One of values from SliderEvent	A slider's event to be registered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback was registered
false	Failed

5.7.8 unregCallback

This method unregisters user's callback for a specified event.

Function prototype

```
bool unregCallback(SliderEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
SliderEvent	state	One of values from SliderEvent	A slider's event to be unregistered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback was unregistered
false	Failed

5.7.9 onInitialTouch

A callback which is invoked when the initial touch flag is enabled and an initial touch event has occurred .

Function prototype

```
virtual void onInitialTouch(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	`position`	0-255	The actual position

Return value

None

5.7.10 onAllRelease

A callback which is invoked when the release flag is enabled and all electrodes assigned to a control has gone to a release state.

Function prototype

```
virtual void onAllRelease(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.7.11 onMovement

A callback which is invoked when the movement flag is enabled and a movement has been detected.

Function prototype

```
virtual void onMovement(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.8 TSSRotary class

The class TSSRotary defines the implementation of the rotary control.

5.8.1 callbackControl

This is the callback handler for the rotary control.

Function prototype

```
virtual void callbackControl(void)
```

Input parameters

None

Return value

None

5.8.2 enable

A method to enable the analog rotary’s control flag.

Function prototype

```
bool enable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of values from TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag was updated
false	Failed

5.8.3 disable

A method to disable the rotary's control flag.

Function prototype

```
bool disable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSControlFlag`	`flag`	One of values from TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag was updated
false	Failed

5.8.4 set

A method to set the rotary's control values.

Function prototype

```
virtual bool set(TSSControlConfig command, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	command	One of values from TSSControlFlag which is valid for the rotary control	TSSControlFlag to be disabled
uint8_t	value	A valid value for rotary control's flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The value was updated
false	Failed

5.8.5 get

A method to retrieve the rotary's control value.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlConfig	config	One of values from TSSControlFlag which is valid for the rotary control	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.8.6 getControlStruct

A method which returns a pointer to the rotary's control structure.

Function prototype

```
const TSS_CSRotary* getControlStruct(void) const
```

Input parameters

None

Return value

A pointer to the rotary's control structure.

5.8.7 regCallback

This method registers user's callback for a specified event.

Function prototype

```
bool regCallback(fRotaryCallback callback, ARotaryEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
fRotary Callback	callback	The pointer to a function	The function to be invoked when the specified state occurs
RotaryEvent	state	One of values from RotaryEvent	The rotary's event to be registered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback was registered
false	Failed

5.8.8 unregCallback

This method unregisters user's callback for a specified event.

Function prototype

```
bool unregCallback(RotaryEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
RotaryEvent	state	One of values from RotaryEvent	A rotary's event to be unregistered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback was unregistered
false	Failed

5.8.9 onInitialTouch

A callback which is invoked when the initial touch flag is enabled and an initial touch event has occurred .

Function prototype

```
virtual void onInitialTouch(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position of a control

Return value

None

5.8.10 onAllRelease

A callback which is invoked when the release flag is enabled and the last touched electrode has switched back to a release state.

Function prototype

```
virtual void onAllRelease(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.8.11 onMovement

A callback which is invoked when a movement flag is enabled and a movement has been detected.

Function prototype

```
virtual void onMovement(uint8_t position)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position	0-255	The actual position

Return value

None

5.9 TSSMatrix class

The class TSSMatrix defines the implementation of the matrix control.

5.9.1 callbackControl

This is the callback handler for the matrix control.

Function prototype

```
virtual void callbackControl(void)
```

Input parameters

None

Return value

None

5.9.2 enable

A method to enable the matrix's control flag.

Function prototype

```
bool enable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	flag	One of values from TSSControlFlag which is valid for the analog rotary control	TSSControlFlag to be enabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag was updated
false	Failed

5.9.3 disable

A method to disable the matrix's control flag.

Function prototype

```
bool disable(TSSControlFlag flag)
```

Input parameters

Type	Name	Valid Range and Values	Description
`TSSControlFlag`	`flag`	One of values from TSSControlFlag which is valid for the matrix control	TSSControlFlag to be disabled

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The flag was updated
false	Failed

5.9.4 set

A method to set the matrix's control values.

Function prototype

```
virtual bool set(TSSControlConfig command, uint8_t value)
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlFlag	command	One of values from TSSControlFlag which is valid for the matrix control	TSSControlFlag to be disabled
uint8_t	value	A valid value for matrix control's flag	The value to be set

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The value was updated
false	Failed

5.9.5 get

A method to retrieve the matrix's control value.

Function prototype

```
virtual uint16_t get(TSSControlConfig config) const
```

Input parameters

Type	Name	Valid Range and Values	Description
TSSControlConfig	config	One of values from TSSControlFlag which is valid for the matrix control	TSSControlFlag to be retrieved

Return value

The return value is uint16_t with the following possible return values:

Return Value	Description
A valid value	A returned value is valid
TSS_ERROR_GETCONF_ILLEGAL_PARAMETER	Failed

5.9.6 getControlStruct

A method which returns a pointer to the matrix's control structure.

Function prototype

```
const TSS_CSMatrix* getControlStruct(void) const
```

Input parameters

None

Return value

A pointer to the rotary's control structure.

5.9.7 regCallback

This method registers user's callback for a specified event.

Function prototype

```
bool regCallback(fMatrixCallback callback, MatrixEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
fMatrixCallback	callback	The pointer to a function	The function to be invoked when the specified state occurs
MatrixEvent	state	One of values from MatrixEvent	The rotary's event to be registered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback was registered
false	Failed

5.9.8 unregCallback

This method unregisters user's callback for a specified event.

Function prototype

```
bool unregCallback(MatrixEvent state)
```

Input parameters

Type	Name	Valid Range and Values	Description
MatrixEvent	state	One of values from MatrixEvent	A matrix's event to be unregistered

Return value

The return value is boolean with the following possible return values:

Return Value	Description
true	The callback was unregistered
false	Failed

5.9.9 onInitialTouch

A callback which is invoked when the initial touch flag is enabled and an initial touch event has occurred .

Function prototype

```
virtual void onInitialTouch(uint8_t position_x, uint8_t position_y)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position_x	0-255	The actual position on x axis
uint8_t	position_y	0-255	The actual position on y axis

Return value

None

5.9.10 onAllRelease

A callback which is invoked when the release flag is enabled and the last touched electrode has switched back to a release state.

Function prototype

```
virtual void onAllRelease(uint8_t position_x, uint8_t position_y)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position_x	0-255	The actual position on x axis
uint8_t	position_y	0-255	The actual position on y axis

Return value

None

5.9.11 onMovement

A callback which is invoked when the movement flag is enabled and a movement has been detected.

Function prototype

```
virtual void onMovement(uint8_t position_x, uint8_t position_y)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	position_x	0-255	The actual gesture’s position x
uint8_t	position_y	0-255	The actual gesture’s position y

Return value

None

5.9.12 onGestures

A callback which is invoked when the gesture flag is enabled and a gesture has been detected.

Function prototype

```
virtual void onGestures(uint8_t ges_distance_x, uint8_t ges_distance_y)
```

Input parameters

Type	Name	Valid Range and Values	Description
uint8_t	ges_distance_x	0-255	A gesture’s distance on x axis
uint8_t	ges_distance_y	0-255	A gesture’s distance on y axis

Return value

None