Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, dsPIC, FlashFlex, flexPWR, Heldo, JukeBloq, KeeLoq, KeeLoq logo, Kleer, LANCheck, LINK MD, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC32 logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, ETHERSYNCH, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and QUIET-WIRE are registered trademarks of Microchip Technology Incorporated in the U.S.A.


SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

gestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2016, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.
ISBN: 978-1-5224-0729-4
Object of Declaration: mTouch® Capacitive Sensing Library for MPLAB® X Code Configurator

EU Declaration of Conformity

Manufacturer: Microchip Technology Inc.
2355 W. Chandler Blvd.
Chandler, Arizona, 85224-6199
USA

This declaration of conformity is issued by the manufacturer.

The development/evaluation tool is designed to be used for research and development in a laboratory environment. This development/evaluation tool is not a Finished Appliance, nor is it intended for incorporation into Finished Appliances that are made commercially available as single functional units to end users under EU EMC Directive 2004/108/EC and as supported by the European Commission's Guide for the EMC Directive 2004/108/EC (8th February 2010).

This development/evaluation tool complies with EU RoHS2 Directive 2011/65/EU.

This development/evaluation tool, when incorporating wireless and radio-telecom functionality, is in compliance with the essential requirement and other relevant provisions of the R&TTE Directive 1999/5/EC and the FCC rules as stated in the declaration of conformity provided in the module datasheet and the module product page available at www.microchip.com.

For information regarding the exclusive, limited warranties applicable to Microchip products, please see Microchip's standard terms and conditions of sale, which are printed on our sales documentation and available at www.microchip.com.

Signed for and on behalf of Microchip Technology Inc. at Chandler, Arizona, USA

Derek Carlson
VP Development Tools

12-Sep-14
Date
Table of Contents

Preface ....................................................................................................................... 8

Chapter 1. Overview
  1.1 Introduction ................................................................................................. 14

Chapter 2. mTouch® Capacitive Sensing Library Module
  2.1 Buttons and Proximity Configuration .......................................................... 17
    2.1.1 Common Button Configuration Overview ........................................... 18
    2.1.1.1 Create New Button ........................................................................ 18
    2.1.1.2 Interface Method ........................................................................... 18
    2.1.1.3 Press Timeout ............................................................................... 19
    2.1.1.4 Negative Capacitance Recovery ................................................... 19
    2.1.1.5 Baseline Filtering ......................................................................... 19
    2.1.2 Individual Button Configuration Overview ......................................... 20
    2.1.2.1 Delete Item .................................................................................. 20
    2.1.2.2 Name ......................................................................................... 20
    2.1.2.3 Sensor ....................................................................................... 20
    2.1.2.4 Threshold ................................................................................... 21
    2.1.2.5 Deviation Scaling ....................................................................... 21
    2.1.3 Common Proximity Sensor Configuration Overview ........................... 22
    2.1.3.1 Create New Proximity Sensor ...................................................... 22
    2.1.3.2 The Interface Method ................................................................... 23
    2.1.3.3 Activity Timeout ......................................................................... 23
    2.1.3.4 Negative Capacitance Recovery ................................................... 23
    2.1.3.5 Baseline Filtering ....................................................................... 23
    2.1.4 Individual Proximity Sensor Configuration Overview ......................... 24
    2.1.4.1 Delete Item .................................................................................. 24
    2.1.4.2 Name ......................................................................................... 24
    2.1.4.3 Sensor ....................................................................................... 24
    2.1.4.4 Threshold ................................................................................... 24
    2.1.4.5 Deviation Scaling ....................................................................... 25
    2.1.6 Hardware Sensor Configuration Overview ........................................... 25
    2.1.6.1 Name ......................................................................................... 26
    2.1.6.2 Pin ............................................................................................. 26
  2.2 AFA (Automatic Frequency Adaptation) ....................................................... 27
    2.2.1 Enabled AFA ...................................................................................... 27
    2.2.2 AFA Filter Timer ............................................................................... 27
Chapter 3. mTouch® Library Module API

3.1 mTouch® Sensor Module ................................................................. 28

3.1.1 enum mtouch_sensor_error MTOUCH_Sensor_Initialize(enum mtouch
sensor_names sensor) ................................................................. 28

3.1.2 enum mtouch_sensor_error MTOUCH_Sensor_Service(enum
mtouch_sensor_names sensor) ..................................................... 28

3.1.3 bool MTOUCH_Sensor_wasSampled(enum mtouch_sensor_names
sensor) ...................................................................................... 28

3.1.4 void MTOUCH_Sensor_Sampled_Reset(void) .................................. 28

3.1.5 void MTOUCH_Sensor_SetSampledCallback(void (*callback)(enum
mtouch_sensor_names sensor)) .................................................. 28

3.1.6 mtouch_sensor_sample_t MTOUCH_Sensor_RawSample_Get(enum
mtouch_sensor_names sensor) ..................................................... 28

3.2 mTouch® Button Module ............................................................... 29

3.2.1 void MTOUCH_Button_Initialize(enum mtouch_button_names button) ... 29

3.2.2 void MTOUCH_Button_Service(enum mtouch_button_names button) ... 29

3.2.3 void MTOUCH_Button_Tick(void) ................................................. 29

3.2.4 void MTOUCH_Button_Recalibrate(void) ..................................... 29

3.2.5 mtouch_button_deviation_t MTOUCH_Button_Deviation_Get(enum
mtouch_button_names button) .................................................. 29

3.2.6 mtouch_button_reading_t MTOUCH_Button_Reading_Get(enum
mtouch_button_names button) .................................................. 29

3.2.7 mtouch_button_reading_t MTOUCH_Button_Baseline_Get(enum
mtouch_button_names button) .................................................. 29

3.2.8 mtouch_button_scaling_t MTOUCH_Button_Scaling_Get(enum
mtouch_button_names button) .................................................. 29

3.2.9 void MTOUCH_Button_Scaling_Set(enum mtouch_button_names button,
mtouch_button_scaling_t scaling) .............................................. 29

3.2.10 mtouch_button_threshold_t MTOUCH_Button_Threshold_Get(enum
mtouch_button_names button) .................................................. 30

3.2.11 void MTOUCH_Button_Threshold_Set(enum mtouch_button_names
button, mtouch_button_threshold_t threshold) ................................ 30

3.2.12 bool MTOUCH_Button_isPressed(enum mtouch_button_names button) .. 30

3.2.13 mtouch_buttonmask_t MTOUCH_Button_Buttonmask_Get(void) ........... 30

3.2.14 void MTOUCH_Button_SetPressedCallback(void (*callback)(enum
mtouch_button_names button)) .............................................. 30

3.2.15 void MTOUCH_Button_SetNotPressedCallback(void (*callback)(enum
mtouch_button_names button)) .............................................. 30

3.3 mTouch® Proximity Module ......................................................... 30

3.3.1 void MTOUCH_Proximity_Initialize(enum mtouch_proximity_names prox) .. 30

3.3.2 void MTOUCH_Proximity_Service(enum mtouch_proximity_names prox) .. 30

3.3.3 void MTOUCH_Proximity_Tick(void) .......................................... 30

3.3.4 void MTOUCH_Proximity_Recalibrate(void) .................................. 30

3.3.5 mtouch_prox_deviation_t MTOUCH_Proximity_Deviation_Get(enum
mtouch_proximity_names prox) .................................................. 31

3.3.6 mtouch_prox_reading_t MTOUCH_Proximity_Reading_Get(enum
mtouch_proximity_names prox) .................................................. 31
3.3.7 mtouch_prox_reading_t MTOUCH_Proximity_Baseline_Get(enum mtouch_proximity_names prox) ............................................................ 31
3.3.8 mtouch_prox_scaling_t MTOUCH_Proximity_Scaling_Get(enum mtouch_proximity_names prox) ............................................................ 31
3.3.9 void MTOUCH_Proximity_Scaling_Set(enum mtouch_proximity_names prox, mtouch_prox_scaling_t scaling) ................................................... 31
3.3.10 mtouch_prox_threshold_t MTOUCH_Proximity_Threshold_Get(enum mtouch_proximity_names prox) ............................................................ 31
3.3.11 void MTOUCH_Proximity_Threshold_Set(enum mtouch_proximity_names prox, mtouch_prox_threshold_t threshold) ............................................ 31
3.3.12 bool MTOUCH_Proximity_isPressed(enum mtouch_proximity_names prox) ...................................................................................................... 31
3.3.13 void MTOUCH_Proximity_SetActivatedCallback(void (*callback)(enum mtouch_proximity_names prox)) ........................................................... 32
3.3.14 void MTOUCH_Proximity_SetNotActivatedCallback(void (*callback)(enum mtouch_proximity_names prox)) ................................................... 32

Chapter 4. Quick Setup for an mTouch® Library Project

4.1 Setting Up the mTouch® Library Project ...................................................... 34
4.2 Adding the ADC, mTouch Hardware Sensor and mTouch Tx Guard Pins ... 35
4.3 Adding a Button and Proximity Sensor .......................................................... 36
4.4 Setting up MyButton ..................................................................................... 37
4.5 Setting up Proximity Sensor ......................................................................... 37
4.6 Configure Interrupt ..................................................................................... 38
4.7 Generating Code ....................................................................................... 38
4.8 Add Code to the Application’s Main Loop .................................................... 38
4.9 Use APIs to Get Button/Proximity State ....................................................... 39
  4.9.1 Polling method ........................................................................................... 39
  4.9.2 Callback Function ...................................................................................... 40
4.10 Use EUSART to Output Data ..................................................................... 41
  4.10.1 Setting up the EUSART Module .............................................................. 41
  4.10.2 Setting up the Code ................................................................................. 41

Worldwide Sales and Service .................................................................................... 42
Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our website (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXXA”, where “XXXXX” is the document number and “A” is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the mTouch® Capacitive Sensing Library Module for MPLAB® X Code Configurator. Items discussed in this chapter include:

• Document Layout
• Conventions Used in this Guide
• This manual uses the following documentation conventions:
  • Recommended Reading
  • The Microchip Website
  • Development Systems Customer Change Notification Service
  • Customer Support
  • Revision History

DOCUMENT LAYOUT

This document describes how to use the mTouch® Capacitive Sensing Library Module for MPLAB® X Code Configurator as a development tool to emulate and debug firmware on a target board, as well as how to program devices. The document is organized as follows:

• Chapter 1. “Overview” – Describes the purpose of the mTouch Capacitive Sensing module and library.
• Chapter 2. “mTouch® Capacitive Sensing Library Module” – Introduces the layout and options for the mTouch module in MCC.
• Chapter 3. “mTouch® Library Module API” – Describes the details of the mTouch library APIs.
• Chapter 4. “Quick Setup for an mTouch® Library Project” – Demonstrates an example of setting up a project using the mTouch library module.
CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

<table>
<thead>
<tr>
<th>DOCUMENTATION CONVENTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Arial font:</strong></td>
</tr>
<tr>
<td>Italic characters</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Initial caps</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Quotes</td>
</tr>
<tr>
<td>Underlined, italic text with right angle bracket</td>
</tr>
<tr>
<td>Bold characters</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>N'Rnnnn</td>
</tr>
<tr>
<td>Text in angle brackets &lt; &gt;</td>
</tr>
<tr>
<td><strong>Courier New font:</strong></td>
</tr>
<tr>
<td>Plain Courier New</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Italic Courier New</td>
</tr>
<tr>
<td>Square brackets [ ]</td>
</tr>
<tr>
<td>Curly brackets and pipe character: { }</td>
</tr>
<tr>
<td>Ellipses...</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
RECOMMENDED READING

This user’s guide describes how to use the mTouch® Capacitive Sensing Library Module for MPLAB® X Code Configurator. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.


This user’s guide provides detailed information about the MPLAB® Code Configurator v3.xx.

For further resources on MCC, refer to:
www.microchip.com/mplab/mplab-code-configurator

THE MICROCHIP WEBSITE

Microchip provides online support via our website at www.microchip.com. This website is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the website contains the following information:

• **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user’s guides and hardware support documents, latest software releases and archived software

• **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing

• **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives
DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip’s customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip website at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

• **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).

• **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators.

• **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICkit™ 3 debug express.

• **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.

• **Programmers** – The latest information on Microchip programmers. These include production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are nonproduction development programmers such as PICSTART® Plus and PICkit 2 and 3.
CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document. Technical support is available through the website at:

http://www.microchip.com/support.

REVISION HISTORY

Revision A (June 2016)

This is the initial release of this document.
Chapter 1. Overview

1.1 INTRODUCTION

The mTouch® Capacitive Sensing Library Module for MPLAB® X Code Configurator allows for quick and easy C code generation of Microchip’s Capacitive Touch button and Proximity sensor solutions.

This library module uses a Graphical User Interface (GUI) to accomplish the following:

• Setting multiple mTouch parameters
• Enabling various mTouch features
• Generating the necessary C code to program onto a PIC® microcontroller

The following chapters of this user’s guide cover each component of the mTouch library module and how to setup a basic mTouch project. Contact your local mTouch representative for additional information or any questions you may have.
Chapter 2. mTouch® Capacitive Sensing Library Module

The mTouch® Capacitive Sensing Library module (Figure 2-1) is split into two sections:

- Buttons and Proximity Configuration
- Automatic Frequency Adaptation (AFA)
2.1 BUTTONS AND PROXIMITY CONFIGURATION

The Button and Proximity Configuration section (Figure 2-2) contains six areas which are accessed by clicking on the items found on the left tree. The six sections are:

1. Common Button Settings
2. Individual Button Settings
3. Common Proximity Sensor Settings
4. Individual Proximity Sensor Settings
5. Hardware Sensor Settings
6. Individual Hardware Sensor Settings

The following sub-chapters provide details about the contents of each area.
2.1.1 Common Button Configuration Overview

FIGURE 2-3: COMMON BUTTON SETTINGS

The Common Button Settings area (Figure 2-3) consists of five sub-areas:
1. Create New Button
2. Interface Method
3. Press Timeout
4. Negative Capacitance Recovery
5. Baseline Filtering

The following sub-chapters provide details about each sub-area.

2.1.1.1 CREATE NEW BUTTON

The Create New Button tab in the Common Button Settings area allows for the creation of an additional button for the mTouch project. When a new button is created, the default name will be Button#, where # will be the lowest number not currently used.

2.1.1.2 INTERFACE METHOD

The mTouch library allows user’s application code to obtain button status in two ways:
- Callback function
- Polling

The callback function method is more efficient than the polling method, but requires additional code space.

Note: The polling method will be available in both options for easy access to the button status.
2.1.1.3 PRESS TIMEOUT

The Press Timeout is a mechanism used to release a stuck button. If a button is pressed for a number of consecutive counts, the button's state will be reset. Each button has its own time-out counter to track the press time.

Time Unit

The underlying press time-out counter will increment each time the MTOUCH_Service_Mainloop() call is performed.

Counts

The counts parameter acts as a threshold to each button’s press time-out counter. Upon the press time-out’s counter exceeding the counts value, the button state will be reset.

2.1.1.4 NEGATIVE CAPACITANCE RECOVERY

Enabling this feature will allow the quick recovery of a button’s baseline if negative capacitance is detected. The negative capacitance means the baseline value is greater than the reading for a button.

Consecutive Decodes

The consecutive decodes parameter specifies the number of consecutive decoding cycles (MTOUCH_Service_Mainloop() call) with negative capacitance that the program has to detect, before resetting the button’s state.

2.1.1.5 BASELINE FILTERING

The baseline of a touch button is the output of a conditionally low-pass filter, and the input of this low pass filter is the button’s raw reading. This allows the baseline to track and react to environmental changes such as temperature and humidity. Upon a button entering a pressed state the baseline will stop updating until released.

Gain

The filter gain parameter determines the baseline filtering strength. The larger the value, the stronger the filter gets, and the more slowly the baseline follows the button reading value.

Update

The update parameter determines how often the system will update the button’s baseline. The larger the value, the less often the system updates the baseline.
2.1.2 Individual Button Configuration Overview

FIGURE 2-4: INDIVIDUAL BUTTON SETTINGS

Each button for an mTouch project receives its own unique settings page. The button’s page (Figure 2-4) consists of five items:

1. Delete the button from the project
2. Name of the button
3. Hardware sensor the button is connected to
4. Threshold for the button
5. Deviation scaling

The following sub-sections provide further details about each component of a button.

2.1.2.1 DELETE ITEM

This button will delete the selected button from the mTouch project. The Delete button is useful when too many buttons have been added to the project and the selected button needs to be removed.

2.1.2.2 NAME

This input field allows users to modify the name of the selected button. An example can be the Play button on a media control panel project. This button can be renamed from Button0 to Play.

2.1.2.3 SENSOR

This selection box allows users to select a hardware sensor to be associated with this button. A hardware sensor can be associated with multiple buttons or proximity sensors. For example, if the user wants to achieve proximity detection and touch detection on the same physical sensor, this sensor can be associated with a button and a proximity sensor.
2.1.2.4 THRESHOLD

The signal deviation is defined as the difference between a button’s reading and the baseline value. The threshold input field allows users to determine the threshold value for signal deviation before the button will be in pressed state.

2.1.2.5 DEVIATION SCALING

The signal deviation will be scaled down from a signed 16-bit integer to a signed 8-bit. If the deviation after scaling down is greater than 127, then the value will be clipped to 127 (0x7F). The deviation scaling parameter controls how many bits will shift to the right. The smaller the value is, the more sensitive the button will get. The goal is to shift the scaled-down deviation to be a value between 64 and 127 when the button is pressed, in order to get the full dynamic range.
2.1.3 Common Proximity Sensor Configuration Overview

The Common Proximity Settings area (Figure 2-5) consists of five sub-areas:

1. Create a new proximity sensor
2. Interface Method
3. Activity Timeout
4. Negative Capacitance Recovery
5. Baseline Filtering

The following sub-sections provide details about each component.

2.1.3.1 CREATE NEW PROXIMITY SENSOR

The Create New Proximity Sensor in the Common Proximity Sensor Settings area allows for the creation of an additional proximity sensor for the mTouch project. When a new proximity sensor is created, the default name will be Proximity#, where # will be the lowest number not currently used.
2.1.3.2 THE INTERFACE METHOD

The mTouch library allows the user’s application code to obtain proximity sensor status in two ways:

- Callback function
- Polling

The callback function method is more efficient than the polling method, but requires additional code space.

**Note:** The polling method will be available in both options for easy access to the proximity sensor status.

2.1.3.3 ACTIVITY TIMEOUT

The Activity Timeout is a mechanism used to release a stuck proximity sensor. If a proximity sensor is pressed for a number of consecutive counts, the proximity sensor’s state will be reset. Each proximity sensor has its own time-out counter to track the press time.

**Time Unit**

The underlying activity time-out counter will increment each time the MTOUCH_Service_Mainloop() call is performed.

**Counts**

The counts parameter acts as a threshold to each proximity sensor’s activity time-out counter. Upon the activity time-out’s counter exceeding the counts value, the proximity sensor’s state will be reset.

2.1.3.4 NEGATIVE CAPACITANCE RECOVERY

Enabling this feature will allow the quick recovery of a bad baseline if negative capacitance is detected. The negative capacitance means the baseline value is greater than the reading for a proximity sensor.

**Consecutive Decodes**

The consecutive decodes parameter specifies the number of consecutive decoding cycles (MTOUCH_Service_Mainloop() call) with negative capacitance that the program has to detect, before resetting the proximity sensor state.

2.1.3.5 BASELINE FILTERING

The baseline of a proximity sensor is the output of a conditionally low-pass filter, and the input of this low-pass filter is the proximity sensor’s raw reading. This allows the baseline to track environmental changes, such as temperature and humidity. Upon a proximity sensor entering an activated state, the baseline will stop updating until released.

**Gain**

The filter gain parameter determines the baseline filtering strength. The larger the value, the stronger the filter gets, and the more slowly the baseline follows the button reading value.

**Update**

The update parameter determines how often the system will update the proximity sensor’s baseline. The larger the value, the less often the system updates the baseline.
2.1.4 Individual Proximity Sensor Configuration Overview

FIGURE 2-6: INDIVIDUAL PROXIMITY SENSOR SETTINGS

Each proximity sensor for an mTouch project receives its own unique settings page (Figure 2-6). The page consists of five items:

1. Delete the proximity sensor from the project
2. Name of the proximity sensor
3. Hardware sensor the proximity sensor is connected to
4. Threshold for the proximity sensor
5. Deviation Scaling

The following sub-sections provide greater detail about each component of a proximity sensor.

2.1.4.1 DELETE ITEM

This button will delete the selected proximity sensor from the mTouch project. The Delete button is useful when too many proximity sensors have been added to the project, and the selected proximity sensor needs to be removed.

2.1.4.2 NAME

This input field allows users to modify the name of the selected proximity sensor. Users can rename the field to something meaningful for their application.

2.1.4.3 SENSOR

This selection box allows users to select a hardware sensor to be associated with this proximity sensor. A hardware sensor may be associated with multiple buttons or proximity sensors.

2.1.4.4 THRESHOLD

The signal deviation for proximity sensor is defined as the difference between a proximity sensor’s reading and its baseline value. To get better sensitivity, the signal deviation is integrated through a low-pass filter and then compared with the threshold value to determine if the proximity sensor will be in an activated state.
2.1.4.5 DEVIATION SCALING

Similarly with a button, the smaller the value, the more sensitive the proximity sensor gets.

2.1.5 Hardware Sensor Configuration Overview

**FIGURE 2-7: COMMON HARDWARE SENSOR SETTINGS**

The Common Hardware Sensor Settings area (Figure 2-7) consists of two items. The first item is displaying the dedicated Tx guard pin, if one has been selected for the project. The other item is to determine the order each hardware sensor will be scanned. To change the scan order of a hardware sensor, first highlight the sensor to move in the Scan Order field list. The next step is to press the Up or Down buttons to change the hardware sensors’ order. The first item in the list will be scanned first, while the last item in the list is scanned last.
2.1.6 Individual Hardware Sensor Configuration Overview

Each hardware sensor receives its own settings page which displays the name of the hardware sensor and the pin the hardware sensor is attached to. The following sub-sections provide greater detail on these two components, as shown in Figure 2-8 above.

2.1.6.1 NAME

This input field allows the user to modify the name of the selected hardware sensor. The user might want to add the package pin number for easy reference to the hardware.

2.1.6.2 PIN

This field allows users to always see which pin the hardware sensor is connected to.
2.2 AFA (AUTOMATIC FREQUENCY ADAPTATION)

The Automatic Frequency Adaptation (AFA) section (Figure 2-9) allows users to enable the advanced noise immunity algorithm. The automatic frequency adaptation routine continuously tracks the amount of noise that is on each sensor, and chooses a new scan frequency intelligently. The routine requires an 8-bit timer to precisely control the scanning frequency. Due to this requirement, mTouch will take ownership of the first available 8-bit timer module, if one is available, when the automatic frequency adaptation routine is enabled. The following sub-sections provide greater detail about the two components of the AFA section.

2.2.1 Enabled AFA

This check-box determines if the automatic frequency adaptation logic will be included when code is generated for the project. The AFA requires extra Flash and RAM memory so this feature may not work with RAM limited controllers.

2.2.2 AFA Filter Timer

This selection box determines which of the available 8-bit timers will be used for the automatic frequency adaptation logic. When a selection is made, the 8-bit timer will be owned by the mTouch library module. This means the timer will no longer be available from the timer peripheral module list. The user will need to enable the timer interrupt in the interrupt manager, as shown in Figure 2-10 below.
Chapter 3. mTouch® Library Module API

The mTouch® library is customized and generated by the MPLAB® X Code Configurator’s Graphical User Interface (GUI). The generated library will be responsible for signal acquisition, noise handling and button/proximity sensor decoding. An application will use the API provided by the mTouch library to obtain button and proximity sensor information. The following sections explain the APIs from sensor, button and proximity modules.

3.1 mTouch® SENSOR MODULE

The mTouch sensor module’s API provides low-level access to the hardware sensor.

3.1.1 enum mtouch_sensor_error MTOUCH_Sensor_Initialize(enum mtouch_sensor_names sensor)

This API function initializes the assigned hardware sensor pin.

3.1.2 enum mtouch_sensor_error MTOUCH_Sensor_Service(enum mtouch_sensor_names sensor)

This API function starts scanning the assigned sensor.

3.1.3 bool MTOUCH_Sensor_wasSampled(enum mtouch_sensor_names sensor)

This API function is used to check if the sensor has been scanned. The sampled status can only be cleared by calling MTOUCH_Sensor_Sampled_Reset().

3.1.4 void MTOUCH_Sensor_Sampled_Reset(void)

This API function is used to clear the sampled status for all the hardware sensors.

3.1.5 void MTOUCH_Sensor_SetSampledCallback(void (*callback)(enum mtouch_sensor_names sensor))

This API function is used to set the callback function that will be called immediately after a sensor has been scanned.

3.1.6 mtouch_sensor_sample_t MTOUCH_Sensor_RawSample_Get(enum mtouch_sensor_names sensor)

This API function is used to get the oversampled sensor reading from the ADC. It is useful if the user wants to implement their own high-level decoding, such as humidity sensor, liquid level sensor, etc.
3.2 mTouch® BUTTON MODULE

The mTouch button module’s API provides access to button data and status information.

3.2.1 void MTOUCH_Button_Initialize(enum mtouch_button_names button)

This API function is used to initialize the assigned button.

3.2.2 void MTOUCH_Button_Service(enum mtouch_button_names button)

This API function is used to service the assigned button, such as determining the button’s status. This function should be called after the associated hardware sensor has been sampled.

3.2.3 void MTOUCH_Button_Tick(void)

This API function is used to increment the pressed time-out counter.

3.2.4 void MTOUCH_Button_Recalibrate(void)

This API function is used to reset and recalibrate all buttons.

3.2.5 mtouch_button_deviation_t MTOUCH_Button_Deviation_Get(enum mtouch_button_names button)

This API function is used to get the signal deviation of the assigned button.

3.2.6 mtouch_button_reading_t MTOUCH_Button_Reading_Get(enum mtouch_button_names button)

This API function is used to get the reading of the assigned button.

3.2.7 mtouch_button_reading_t MTOUCH_Button_Baseline_Get(enum mtouch_button_names button)

This API function is used to get the baseline of the assigned button.

3.2.8 mtouch_button_scaling_t MTOUCH_Button_Scaling_Get(enum mtouch_button_names button)

This API function is used to get the scaling factor of the assigned button.

3.2.9 void MTOUCH_Button_Scaling_Set(enum mtouch_button_names button, mtouch_button_scaling_t scaling)

This API function is used to set the scaling factor of the assigned button.

Note: The scaling factor set using this API will be lost after a power cycle. The scaling factor’s permanent value may be changed through the MPLAB® X Code Configurator mentioned in Section 2.1.2.5 “Deviation Scaling” or through the initialization code prior to compiling
3.2.10 mtouch_button_threshold_t
MTOUCH_Button_Threshold_Get(enum mtouch_button_names button)

This API function is used to get the deviation threshold of the assigned button.

3.2.11 void MTOUCH_Button_Threshold_Set(enum mtouch_button_names button, mtouch_button_threshold_t threshold)

This API function is used to set the deviation threshold of the assigned button. Similar to the MTOUCH_Button_Scaling_Set function, this change of threshold is only temporary.

3.2.12 bool MTOUCH_Button_isPressed(enum mtouch_button_names button)

This API function is used to get the current button state of the assigned button.

3.2.13 mtouch_buttonmask_t
MTOUCH_Button_Buttonmask_Get(void)

This API function is used to get the current button state of all buttons.

3.2.14 void MTOUCH_Button_SetPressedCallback(void (*callback)(enum mtouch_button_names button))

This API function is used to set the user’s callback function for a button press event.

3.2.15 void MTOUCH_Button_SetNotPressedCallback(void (*callback)(enum mtouch_button_names button))

This API function is used to set the user’s callback function for a button release event.

3.3 mTouch® PROXIMITY MODULE

The mTouch proximity module API provides access to proximity sensor data and status information.

3.3.1 void MTOUCH_Proximity_Initialize(enum mtouch_proximity_names prox)

This API function is used to initialize the assigned proximity sensor.

3.3.2 void MTOUCH_Proximity_Service(enum mtouch_proximity_names prox)

This API function is used to service the assigned proximity sensor, such as determining its status. This function should be called after the associated sensor has been sampled.

3.3.3 void MTOUCH_Proximity_Tick(void)

This API function is used to increment the activation time-out counter.

3.3.4 void MTOUCH_Proximity_Recalibrate(void)

This API function is used to reset and recalibrate all proximity sensors.
3.3.5  mtouch_prox_deviation_t
   MTOUCH_Proximity_Deviation_Get(enum mtouch_proximity_names prox)

   This API function is used to get the signal deviation of the assigned proximity sensor.

3.3.6  mtouch_prox_reading_t
   MTOUCH_Proximity_Reading_Get(enum mtouch_proximity_names prox)

   This API function is used to get the reading of the assigned proximity sensor.

3.3.7  mtouch_prox_reading_t
   MTOUCH_Proximity_Baseline_Get(enum mtouch_proximity_names prox)

   This API function is used to get the baseline of the assigned proximity sensor.

3.3.8  mtouch_prox_scaling_t
   MTOUCH_Proximity_Scaling_Get(enum mtouch_proximity_names prox)

   This API function is used to get the scaling factor of the assigned proximity sensor.

3.3.9  void MTOUCH_Proximity_Scaling_Set(enum mtouch_proximity_names prox, mtouch_prox_scaling_t scaling)

   This API function is used to set the scaling factor of the assigned proximity sensor.

   Note: The scaling factor that is set by using this function will be lost after a power cycle. The scaling factor’s permanent value may be changed through the MPLAB® X Code Configurator or through the initialization code prior to compiling.

3.3.10 mtouch_prox_threshold_t
   MTOUCH_Proximity_Threshold_Get(enum mtouch_proximity_names prox)

   This API function is used to get the deviation threshold of the assigned proximity sensor.

3.3.11 void MTOUCH_Proximity_Threshold_Set(enum mtouch_proximity_names prox, mtouch_prox_threshold_t threshold)

   This API function is used to set the deviation threshold of the assigned proximity sensor. Similar to the MTOUCH_Proximity_Scaling_Set function, this change of threshold is only temporary.

3.3.12 bool MTOUCH_Proximity_isPressed(enum mtouch_proximity_names prox)

   This API function is used to get the current state of the assigned proximity sensor.
3.3.13  void MTOUCH_Proximity_SetActivatedCallback(void (*callback)(enum mtouch_proximity_names prox))
This API function is used to set the user’s callback function for a proximity activation event.

3.3.14  void MTOUCH_Proximity_SetNotActivatedCallback(void (*callback)(enum mtouch_proximity_names prox))
This API function is used to set the user’s callback function for a proximity release event.
Chapter 4. Quick Setup for an mTouch® Library Project

This chapter provides step-by-step instruction for creating and programming a basic mTouch® project onto a PIC® microcontroller. A PIC16F1619 is used for this example project, but the same steps will work with most mTouch compatible PIC microcontrollers. This example project includes the following:

- One button called MyButton
- One proximity sensor called MyProximity
- An ADC pin enabled
- AFA enabled

The button and proximity sensor also have to use separate hardware sensors.

4.1 SETTING UP THE mTouch® LIBRARY PROJECT

The first step of a MCC configuration is setting up the system module, as shown in Figure 4-1. For optimal performance, a fast system clock is recommended. In this example, the system clock is set to 32 MHz.

**FIGURE 4-1: SYSTEM MODULE SETTING**

![System Module Setting](image)

The next step for this example project is to add the ADC and the mTouch library module to the Project Resources area, as indicated in Figure 4-2.

**FIGURE 4-2: PROJECT RESOURCES**

![Project Resources](image)
To add the ADC module, navigate to the ADC item in the Device Resources panel, as shown in Figure 4-3 above. Click ADC such that the arrow to the left points downward. Select the ADC that does not have an arrow next to the ADC text. After the ADC has been selected it will be removed from the Device Resources list and it will be added into the Project Resources list. The center pane should also display the ADC module.

The final part of this step is to add the mTouch library module. Navigate to the mTouch item in the Device Resources panel, as indicated in Figure 4-4. At this point, double clicking **mTouch** will add it to the Project Resources. The next step is enabling all of the pins for the example project.

### 4.2 ADDING THE ADC, mTouch HARDWARE SENSOR AND mTouch TX GUARD PINS

The above diagram in Figure 4-5 displays the completed pin layout for this example project. The following components need to be enabled: three ADC pins, two mTouch hardware sensor pins and an mTouch Tx guard pin.

The first step is to enable the ADC pins for this project. For this example, pins RA1, RA4 and RC0 will be used. To enable the ADC pins, click on the respective lock that is in the unlocked position. When clicked, the blue lock will switch to a green background and the icon will be in a locked position. When an icon has a green background and a locked icon it indicates the pin is enabled.
After enabling the first ADC pin the mTouch module will appear in the Pin Mapper pane with a row called Sensor. After RA1 and RA4 have been enabled for the ADC module lock, the respective mTouch sensor pins will be enabled as well. After locking the mTouch sensor pins there will be a chain icon displayed instead of the locked icon. The chain icon means that two modules have linked ownership with the pin's functionality.

Upon linking the two mTouch hardware sensors with ADC pins, it is time to assign a TX guard pin. The Tx guard assists in boosting the amount of signal for buttons and proximity sensors. For this example project the Tx guard will be assigned to pin RB6. After the PIC microcontroller’s pins are setup correctly, it is time to add the single button and single proximity sensor to the project.

### 4.3 ADDING A BUTTON AND PROXIMITY SENSOR

The next step for the mTouch project example is to create a button and proximity sensor. The mTouch module needs to be shown in the center pane. If another module is currently shown, go into the Project Resources area and click on the mTouch library module. Figure 4-6 shows how to add a new button. The Button and Proximity Configuration section needs to be expanded and the Common Button Settings pane needs to be displayed. Once the Common Button Settings pane is shown, click Create New Button in the upper right corner. After this button has been clicked, a Button0 will appear in the mTouch tree. To add a new proximity sensor, go to the Common Proximity Sensor Settings pane and click the Create New Proximity Sensor button. Upon completing this set the project will now have one button and one proximity sensor. The next steps involve setting up the button and proximity sensor.

**FIGURE 4-6: ADDING A NEW BUTTON**

![mTouch module configuration](image-url)
4.4 SETTING UP MyBUTTON

For this project the button is to be renamed “MyButton”. To accomplish this step, click the **Button0: No sensor selected** in the mTouch tree, as indicated in Figure 4-7. In the Name field for this button’s settings change the text from “Button0” to “MyButton”. For the Sensor selection box select the “Sensor_AN1: RA1 / AN1” option. MyButton is now connected to the hardware sensor on the RA1 / AN1 pin. The next step is to setup the proximity sensor.

**FIGURE 4-7: SETUP MyBUTTON**

![Diagram showing Button and Proximity Configuration with MyButton settings](image)

4.5 SETTING UP PROXIMITY SENSOR

For this project, the proximity sensor is to be renamed “MyProximity”. To accomplish this step first, click the “Proximity0: No sensor selected” option in the mTouch tree, as shown in Figure 4-8 below. In the name field for this proximity sensor's settings change the text from “Proximity0” to “MyProximity”. For the Sensor selection box select the “Sensor_AN3: RA4 / AN3” option. This means MyProximity has been connected to the hardware sensor on the RA4 / AN3 pin.

**FIGURE 4-8: SETUP MyPROXIMITY**

![Diagram showing Button and Proximity Configuration with MyProximity settings](image)
4.6 CONFIGURE INTERRUPT

![Interrupt Module Diagram]

If the AFA feature is enabled, then the interrupt of the selected 8-bit timer also needs to be enabled. To keep the timer’s interrupt as constant as possible, the selected 8-bit timer needs to be the first item in the interrupt vector’s list. Figure 4-9 displays the selected AFA timer as the first item in the interrupt vector list.

4.7 GENERATING CODE

At this point everything in the MPLAB® X Code Configurator’s project is complete. To recap, the mTouch module has a button and a proximity sensor and each of them are connected to a unique hardware sensor, and the AFA feature is enabled. To generate code, click the Generate button above the center pane. After the code has been generated there is one last step before the final code may be compiled and programmed onto the PIC® microcontroller.

4.8 ADD CODE TO THE APPLICATION’S MAIN LOOP

The mTouch routine needs to be continuously called from the main loop, and the globe and peripheral interrupt must be enabled for timer if AFA is enabled, as shown in Figure 4-10 below.

```c
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();
    
    while (1)
    {
        MTOUCH_Service_Mainloop();
        // Add your application code
    }
}
```

At this point, the mTouch configuration is complete.
4.9 USE APIs TO GET BUTTON/PROXIMITY STATE

There are two ways to get the button/proximity sensor state:

• Polling Method
• Callback Function

4.9.1 Polling method

The mTouch button/proximity sensor provides the API to get the state for a particular button/proximity sensor. Users can call this API at any time to obtain the state, as shown in Figure 4-11 below.

FIGURE 4-11: POLLING BUTTON0’S STATE

```c
while (1) {
    MTOUCH_Service_Mainloop();

    if (MTOUCH_Button_isPressed(Button0) == true) {
        //button press code
    } else {
        //button release code
    }
    // Add your application code

```
4.9.2 Callback Function

The callback function method requires the users to create a function to handle the press/touch event, and then pass the function pointer to the callback setter function, as shown in Figure 4-12 below.

**FIGURE 4-12: CALLBACK FUNCTION TO HANDLE BUTTON PRESS EVENT**

```c
#include "mcc.h"
#include "stdio.h"
#include "mtouch/mtouch_button.h" // Include the button header file

// Main application

void main(void) {
    // initialize the device
    SYSTEM_Initialize();

    // When using interrupts, you need to set the Global and Peripheral Interrupt
    // Use the following macros to:

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();

    // Disable the Global Interrupts
    INTERRUPT_GlobalInterruptDisable();

    // Disable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptDisable();

    // Setup your callback function for mTouch button state change
    MTOUCH_Button_SetPressedCallback(myButton_pressedCallback);
    MTOUCH_Button_SetNotPressedCallback(myButton_NotPressedCallback);

    while (1) {
        MTOUCH_Service_Mainloop();
    }
}

void myButton_pressedCallback(enum mtouch_button_names button) {
    // application code that handles button press
}

void myButton_NotPressedCallback(enum mtouch_button_names button) {
    // application code that handles button release
}
```
4.10 USE EUSART TO OUTPUT DATA

It is always recommended to monitor the mTouch data in real-time for parameter tuning. In this example, a EUSART-based debugging interface will be used.

4.10.1 Setting up the EUSART Module

The first step is to load the EUSART module from the device resources, selecting Asynchronous mode. The MPLAB XC8 compiler has implemented the `printf` as a library function to format string and send to a destination. To use the `printf` function and send the string to EUSART, the EUSART module needs to be redirected to STDIO, as shown in Figure 4-13 below.

**FIGURE 4-13: SETTING UP THE EUSART MODULE**

4.10.2 Setting up the Code

Once the EUSART code is generated, the user will be able to output any variables and mTouch data as ASCII format via the serial communication interface. Any terminal software (HyperTerm, CoolTerm) will be able to display the data in real-time on a PC. In this example, the deviation of MyButton and MyProximity will be output onto the EUSART TX pin.

**FIGURE 4-14: OUTPUT MYBUTTON AND MYPROXIMITY DATA THROUGH EUSART**

```c
while (1) {
    MTOUCH_Service_Mainloop();
    printf("%d,%d",MTOUCH_Button_Deviation_Get(MyButton),MTOUCH_Proximity_Deviation_Get(MyProximity));
    // Add your application code
}
```