*// @file main.c  
// @brief Main logic for Battery+  
//  
// Contains the higher level logic code  
//  
// @author Eric D. Phillips  
// @date November 22, 2015  
// @bugs No known bugs*#include **<pebble.h>**#include **"drawing.h"***// Main constants*#define **MENU\_CELL\_COUNT** 1  
#define **MENU\_CELL\_HEIGHT\_TALL** 65  
  
*// Main data structure***static struct** {  
 Window \*window; *//< The base window for the application* MenuLayer \*menu; *//< The main menu layer for the application*} main\_data;  
  
  
*// MenuLayer row draw callback***static void** prv\_menu\_row\_draw\_handler(GContext \*ctx, **const** Layer \*layer, MenuIndex \*index,  
 **void** \*context) {  
 drawing\_render\_cell(ctx); *// <-- This gets called and goes to drawing.c*}  
  
*// MenuLayer get row count callback***static** uint16\_t prv\_menu\_get\_row\_count\_handler(MenuLayer \*menu, uint16\_t index, **void** \*context) {  
 **return MENU\_CELL\_COUNT**;  
}  
  
*// MenuLayer get row height callback***static** int16\_t prv\_menu\_get\_row\_height\_handler(MenuLayer \*menu, MenuIndex \*index, **void** \*context) {  
 **return MENU\_CELL\_HEIGHT\_TALL**;  
}  
  
  
*// Initialize menu layer***static void** prv\_initialize\_menu\_layer(Layer \*window\_root, GRect window\_bounds) {  
 main\_data.menu = menu\_layer\_create(window\_bounds);  
 menu\_layer\_set\_highlight\_colors(main\_data.menu, **GColorChromeYellow**, **GColorBlack**);  
 menu\_layer\_set\_center\_focused(main\_data.menu, **true**);  
 menu\_layer\_set\_callbacks(main\_data.menu, **NULL**, (MenuLayerCallbacks) {  
 .draw\_row = prv\_menu\_row\_draw\_handler,  
 .get\_num\_rows = prv\_menu\_get\_row\_count\_handler,  
 .get\_cell\_height = prv\_menu\_get\_row\_height\_handler,  
 });  
 menu\_layer\_set\_click\_config\_onto\_window(main\_data.menu, main\_data.window);  
 layer\_add\_child(window\_root, menu\_layer\_get\_layer(main\_data.menu));  
}  
  
*// Initialize the program***static void** prv\_initialize(**void**) {  
 *// initialize window* main\_data.window = window\_create();  
 Layer \*window\_root = window\_get\_root\_layer(main\_data.window);  
 GRect window\_bounds = layer\_get\_bounds(window\_root);  
 window\_stack\_push(main\_data.window, **true**);  
 *// initialize menu layer* prv\_initialize\_menu\_layer(window\_root, window\_bounds);  
}  
  
*// Entry point***int** main(**void**) {  
 prv\_initialize();  
 app\_event\_loop();  
}

*//! @file drawing.h  
//! @brief Main drawing code  
//!  
//! Contains all the drawing code for this app.  
//!  
//! @author Eric D. Phillips  
//! @date November 28, 2015  
//! @bugs No known bugs*#pragma once  
#include **<pebble.h>***//! Render a MenuLayer cell  
//! @param ctx The cell's drawing context***void** drawing\_render\_cell(GContext \*ctx);

*// @file drawing.c  
// @brief Main drawing code  
//  
// Contains all the drawing code for this app.  
//  
// @author Eric D. Phillips  
// @date November 28, 2015  
// @bugs No known bugs*#include **"drawing.h"***// Clock cell constants*#define **CELL\_CLOCK\_TICK\_WIDTH** 3  
#define **CELL\_CLOCK\_TICK\_LENGTH** 8  
#define **CELL\_CLOCK\_HR\_HAND\_INSET** 37  
#define **CELL\_CLOCK\_MIN\_HAND\_INSET** 23  
  
  
**static void** prv\_print\_mem(**void** \*ptr, size\_t length) {  
 **printf**(**"Start print [%d]..."**, (**int**)length);  
 **for** (uint8\_t \*byte = ptr; byte < (uint8\_t\*)ptr + length; byte++) {  
 **printf**(**"%d"**, \*byte);  
 }  
}  
  
*// Render rich text with different fonts  
// Arguments are specified as two char\* arrays, one of text and the other of fonts***static void** prv\_render\_rich\_text(GContext \*ctx, GRect bounds, **char** \*\*text, **char** \*\*font) {  
 *// print arguments* **printf**(**"Bounds: %d %d %d %d"**, bounds.origin.x, bounds.origin.y, bounds.size.w,  
 bounds.size.h);  
 **printf**(**"Text: %s"**, text[0]);  
 **printf**(**"Font: %s"**, font[0]);  
 psleep(200);  
 *// draw simple geometry to test if GContext is valid* graphics\_fill\_circle(ctx, grect\_center\_point(&bounds), 30);  
 graphics\_draw\_rect(ctx, grect\_inset(bounds, **GEdgeInsets1**(10)));  
 **printf**(**"Simple Draw Complete"**);  
 psleep(200);  
 *// create text draw args* **char** my\_buff[] = **"Hello"**;  
 GRect my\_bounds = **GRect**(0, 0, 144, 50);  
 GFont my\_font = fonts\_get\_system\_font(**FONT\_KEY\_GOTHIC\_14**);  
 **printf**(**"Attributes Created"**);  
 psleep(200);  
 *// draw text* graphics\_draw\_text(ctx, my\_buff, my\_font, my\_bounds, ***GTextOverflowModeFill***, ***GTextAlignmentCenter***,  
 **NULL**); *// <-- Crashes here* **printf**(**"Text Draw Complete"**);  
 psleep(200);  
}  
  
  
*// -------- THIS FUNCTION IS NEVER CALLED ---------- //  
// However, changing anything here still prevents the crash***static void** uncalled\_function\_1(**void**) {  
 *// get time* time\_t t\_time;  
 t\_time = time(**NULL**);  
 tm tm\_time;  
 tm\_time = \*localtime(&t\_time);  
 **printf**(**"Clock 1"**);  
*// check draw mode* **char** digit\_buff[6], symbol\_buff[3], date\_buff[16];  
 *// get text* strftime(digit\_buff, **sizeof**(digit\_buff), **"%l:%M"**, &tm\_time);  
 strftime(symbol\_buff, **sizeof**(symbol\_buff), **"%p"**, &tm\_time);  
 strftime(date\_buff, **sizeof**(date\_buff), **"%a, %b %e"**, &tm\_time);  
 *// draw text* **char** \*txt[2];  
 txt[0] = digit\_buff;  
 txt[1] = symbol\_buff;  
 **char** \*font[2];  
 font[0] = **FONT\_KEY\_GOTHIC\_18\_BOLD**;  
 font[1] = **FONT\_KEY\_GOTHIC\_18\_BOLD**;  
 prv\_render\_rich\_text(**NULL**, **GRectZero**, txt, font);  
 **char** \*txt\_1[5];  
 txt\_1[0] = date\_buff;  
 **char** \*font\_1[5];  
 font\_1[0] = **FONT\_KEY\_GOTHIC\_18\_BOLD**;  
 prv\_render\_rich\_text(**NULL**, **GRectZero**, txt\_1, font\_1);  
 *// draw tick marks* GRect tick\_bounds;  
 tick\_bounds = grect\_inset(**GRectZero**, **GEdgeInsets1**(-15));  
 graphics\_context\_set\_stroke\_width(**NULL**, **CELL\_CLOCK\_TICK\_WIDTH**);  
 graphics\_context\_set\_stroke\_color(**NULL**, **GColorWhite**);  
 int32\_t angle;  
 **for** (angle = 0; angle < **TRIG\_MAX\_ANGLE**; angle += **TRIG\_MAX\_ANGLE** / 12) {  
 graphics\_draw\_line(**NULL**, gpoint\_from\_polar(tick\_bounds, ***GOvalScaleModeFillCircle***, angle),  
 grect\_center\_point(&tick\_bounds));  
 }  
 graphics\_fill\_rect(**NULL**, grect\_inset(**GRectZero**, **GEdgeInsets1**(**CELL\_CLOCK\_TICK\_LENGTH**)), 0,  
 ***GCornerNone***);  
 **printf**(**"Clock 3"**);  
 *// draw date* GRect date\_bounds;  
 date\_bounds = **GRectZero**;  
 date\_bounds.origin.y += date\_bounds.size.h / 2 - 12;  
 date\_bounds.origin.x += date\_bounds.size.w \* 2 / 3;  
 date\_bounds.size.w /= 4;  
 strftime(date\_buff, **sizeof**(date\_buff), **"%e"**, &tm\_time);  
 graphics\_draw\_text(**NULL**, date\_buff, fonts\_get\_system\_font(**FONT\_KEY\_GOTHIC\_18\_BOLD**), date\_bounds,  
 ***GTextOverflowModeFill***, ***GTextAlignmentCenter***, **NULL**);  
 *// calculate hands* GPoint hr\_point = gpoint\_from\_polar(grect\_inset(**GRectZero**, **GEdgeInsets1**(**CELL\_CLOCK\_HR\_HAND\_INSET**)),  
 ***GOvalScaleModeFillCircle***,  
 (tm\_time.tm\_hour % 12 \* 60 + tm\_time.tm\_min) \* **TRIG\_MAX\_ANGLE** / (1440 / 2));  
 GPoint min\_point = gpoint\_from\_polar(grect\_inset(**GRectZero**,  
 **GEdgeInsets1**(**CELL\_CLOCK\_MIN\_HAND\_INSET**)), ***GOvalScaleModeFillCircle***,  
 tm\_time.tm\_min \* **TRIG\_MAX\_ANGLE** / 60);  
 *// draw hands* graphics\_draw\_line(**NULL**, grect\_center\_point(&**GRectZero**), **GPointZero**);  
 graphics\_draw\_line(**NULL**, grect\_center\_point(&**GRectZero**), **GPointZero**);  
}  
  
  
*// Render a MenuLayer cell***void** drawing\_render\_cell(GContext \*ctx) {  
 *// get cell parameters* **char** \*txt[5];  
 txt[0] = **"Hello"**;  
 txt[1] = **"World"**;  
 **char** \*font[5];  
 font[0] = **FONT\_KEY\_GOTHIC\_18\_BOLD**;  
 font[1] = **FONT\_KEY\_GOTHIC\_18\_BOLD**;  
 *// render cell* prv\_render\_rich\_text(ctx, **GRect**(0, 0, 144, 50), txt, font); *// <-- Enters here  
  
  
 // --------- FROM HERE ON NEVER GETS CALLED ---------- //  
 // However, changing anything here still prevents the crash* **if** (rand() < 0) {  
 uncalled\_function\_1();  
 }  
}