

Microchip Google PowerMeter Reference Implementation Help

Table of Contents

Introduction	1
Additional Information	1
Getting Help	2
SW License Agreement	3
Running the Demos	4
Firmware Requirements	4
Development Tools	4
Configuration and Programming	6
Authentication	7
Operation	8
Using the Code	10
How the Application Works	10
How the Web Page Works	10
Creating a Custom Application	11
Application API	14
Functions	14
CustomHTTPApp	14
HTTPDeactivateDevice Function	14
HTTPPostAuthenticate Function	15
HTTPPostAuthInfo Function	15
ginsu	16
CalculatePubKeyHash Function	17
ConvertUtcSecToString Function	17
GetBuildNumber Function	18
GetCumulativePower Function	18
GetHashListEntry Function	19
GetIpv4 Function	19
GetMac Function	20
GetPreferenceInt32 Function	20
GetPreferenceStr Function	21

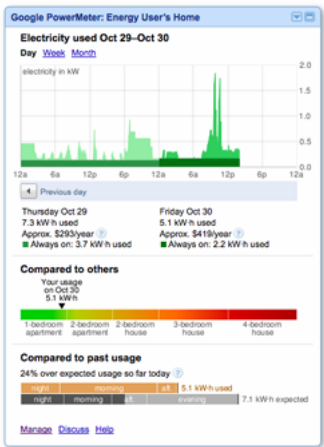
GetSQ Function	21
GetSQDetails Function	22
GetUptimeSec Function	22
GetUtcTimeSec Function	23
GetWebPort Function	23
GInitForWork Function	24
GPollForWork Function	24
GSendInit Function	24
GUpdateNVRam Function	25
PadBuffer Function	25
SensorStillActive Function	26
SetHashListEntry Function	26
SetupCalculatePubKeyHash Function	27
MainDemo	27
InitNVMemContents Function	27
ProcessIO Function	28
SaveNVMemContents Function	28
SavePowerMeterPreferences Function	29
Structures	29
G_PKEY_INFO Structure	30
GSTATUS_COUNTERS Enumeration	30
NV_MEM_STRUCTURE_OFFSETS Enumeration	31
PMButtonState Enumeration	32
PMCaptureMode Enumeration	32
POWER_METER_PREFERENCES Structure	32
PREF_LIST Enumeration	33
Macros	33
APPLICATION_BUILD Macro	34
DEVICE_MANUFACTURER Macro	34
DEVICE_MODEL Macro	34
DEVICE_NUM_SENSORS Macro	35
GOOGLE_DATA_PORT Macro	35
GOOGLE_HOST Macro	35
GOOGLE_STATUS_PORT Macro	35
GQ_NUM_QUEUES Macro	36
MAX_NUM_PUBKEY_HASH Macro	36
PUBKEY_HASH_SIZE Macro	36
Variables	36
AppConfig Variable	37
auth_path Variable	37
auth_token Variable	38

captureMode Variable	38
gActivationConfirm Variable	38
gActivationString1 Variable	38
gActivationString2 Variable	39
gActivationString3 Variable	39
gActivationString4 Variable	39
gActivationString5 Variable	39
gActivationString6 Variable	40
gCumulativePower Variable	40
goog_host Variable	40
gpkey_info Variable	40
gPowerMeterPreferences Variable	41
StackStartTime Variable	41

Index

a

1 Introduction



The Google PowerMeter service provides the ability to view power usage data through a web-based interface. This data can be uploaded to Google by embedded devices in breaker boxes, power strips, or even in electronics themselves. Tracking power consumption will make users aware of the direct effect that using their devices has on their power bills, which will allow them to adjust their power usage to lower their bills and avoid using devices during peak power consumption hours. This makes the ability to monitor power consumption an attractive feature in a new device.

These demos and this code are intended to ease the addition of Google PowerMeter support to your devices. The TCP/IP Stack Help file (installed with the Microchip Application Libraries) gives a detailed description of Microchip's TCP/IP Stack. The code files provided by Google include:

File	Function
ginsu.c/h	Provides an interface between the Google reference code and Microchip's TCP/IP Stack. Contains functions that the user may modify to customize his or her application (see page 11).
gcapture.c/h	Implements functions to determine when and how to capture power readings.
gcounter.c/h	Implements state counters, status counters, and timestamps used by the reference code. Provides functions to get/set them.
gpubkey.c/h	Implements a system to store and compare hashes of server certificates' public keys. Comparing stored hashes to hashes of a received certificate's key provides a workaround for the TCP/IP Stack's lack of server certificate validation. This prevents man-in-the-middle attacks.
gqueue.c/h	Implements a queue to store power reading values and timestamps.
gsend.c/h	Provides HTTP Client functionality to transmit power readings to the Google server.
gstatus.c/h	Provides HTTP Client functionality to transmit device status updates to the Google server. This information is not viewable by the user as of March 15, 2010.
gutility.c/h	Implements utility functions for the Google reference code.

1.1 Additional Information

Where to find additional information about Google PowerMeter.

Description

The following links can provide you with more information about Google PowerMeter:

- [Google PowerMeter API Documentation](#) -- Documentation & supporting material (client libraries, FAQs etc.) on the Google PowerMeter API.
 - [Google PowerMeter Description](#) -- A brief, non-technical description of how the Google PowerMeter service can help conserve energy.
 - [Why Did Google Create PowerMeter?](#) -- Answers to this and other questions that your customers will probably ask.
-

1.2 Getting Help

Where to get additional help with this demo and with Google PowerMeter.

Description

For additional help regarding any Microchip Technology products, including MPLAB IDE, the C18/C30/C32 compilers, the Microchip TCP/IP Stack, or the demo hardware or firmware you can submit ticket requests at <http://support.microchip.com>.

For technical questions about Google PowerMeter or the Google PowerMeter Reference Code that can't be answered by the sites in the Additional Information section, please email google-powermeter-partner-support@google.com.

For questions about Google PowerMeter contract, product, and legal issues, please contact powermeter-partner@google.com.

2 SW License Agreement

Describes the licensing and terms of use of this distribution.

Description

The files in this demonstration are governed by multiple software license agreements.

Google Software License Agreement

All code made available by Google, Inc. (the code contained in `gcapture.c`, `gcapture.h`, `gpubkey.c`, `gpubkey.h`, `gqueue.c`, `gqueue.h`, `gsend.c`, `gsend.h`, `gstatus.c`, `gstatus.h`, `gutility.c`, `gutility.h`, and `ginsu.h`) is licensed under the Apache License, version 2.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at:

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Microchip Technology Software License Agreement

The code made available by Microchip Technology, Inc. is governed under the following license agreement:

Microchip Demo Code for Google PowerMeter. Copyright 2010 Microchip Technology Inc. and its licensors. All rights reserved.

Microchip licenses to you the right use, modify, copy, and distribute the accompanying Microchip demo code only when used with or embedded on a Microchip microcontroller or Microchip digital signal controller that is integrated into your product or a third party product. Any redistributions of Microchip's demo code in compliance with the foregoing must include a copy of this entire notice.

THE MICROCHIP DEMO CODE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

THIRD PARTY SOFTWARE: Notwithstanding anything to the contrary, any third party software accompanying this software - including but not limited to Google's reference code - is subject to the terms and conditions of license agreement of such third party, such as the Apache License Version 2.0 (www.apache.org/licenses/). To the extent required by such third party licenses, the terms of the third party license will apply in lieu of the terms provided herein. To the extent the terms of such third party licenses prohibit any of the restrictions described herein, such restrictions will not apply to such third party software. THIRD PARTY SOFTWARE IS SUBJECT TO THE FOREGOING WARRANTY DISCLAIMER AND LIMIT ON LIABILITY PROVIDED IN THE PARAGRAPH ABOVE.

3 Running the Demos

Describes how to set up and run the demonstration projects.

Description

This section describes how to set up and run the demonstration projects.

3.1 Firmware Requirements

Describes the firmware required for this demo, and how to install it.

Description

You will require two firmware packages to run this demo. Each of these must be installed in the order listed, as some files will be overwritten.

1. The Microchip Application Libraries, including TCP/IP Stack v5.25 or later. This code is available at www.microchip.com/mal.
2. Microchip's Data Encryption Libraries. This demo uses the SSL security layer to communicate with Google. To use SSL with the TCP/IP stack, you will require these libraries. They are available in a CD-based or downloadable format from [MicrochipDirect](#) for a nominal \$5 charge (required to comply with U.S. cryptographic export restriction screening). You must execute the "Microchip TCPIP Stack vX.XX Encryption Add-on.exe" installer to install the files required by the demo. The ARCFOUR.c/h and RSA.c/h cryptographic files in this installer will replace the dummy versions found with the default TCP/IP Stack installation.

3.2 Development Tools

Describes the hardware and software tool that are required to run these demos.

Description

These projects are configured to use either the **Explorer 16** development board (part number [DM240001](#)) or the **dsPIC33E/PIC24E USB Starter Kit + IO Expansion board**.

You will also need one of the following PICtail Plus daughter boards to provide TCP/IP connectivity:

- Ethernet PICtail Plus Daughter Board for 10Mbps ([AC164123](#))
- Fast 100Mbps Ethernet PICtail Plus Daughter Board ([AC164132](#))
- WiFi® PICtail Plus Daughter Board ([AC164136-2](#)).

The demo projects that use the Explorer 16 development board require one of the following plug-in modules (PIMs). Note that the Explorer 16 kit listed includes the PIC24F Plug-In Module (PIM).

- PIC24F 100P to 100P TQFP Plug-In Module with PIC24FJ128GA010 ([MA240011](#))
- PIC32MX 100P to 100P Plug-In Module with PIC32MX360F512L ([MA320001](#))

If you are using the Google PowerMeter EZConfig Demo, you will require the Explorer 16 and the WiFi PICtail Plus Daughter Board.

If you are using the Energy Monitoring Demo, you will require the Explorer 16, [PIC18F87J72 Energy Monitoring PICtail Plus Daughter Board](#), the PIC24F PIM, and one of the listed TCP/IP PICtails.

If you are using the dsPIC33E/PIC24E USB Starter Kit, you will require the Fast 100 Mbps Ethernet PICtail Plus Daughter Board.

For information on how to connect these boards, please consult the TCP/IP Stack Help File, installed with the Microchip Application Libraries.

Demo Compatibility Table

The following configurations are supported by default in these demos (some configuration (see page 6) may be necessary).

Google PowerMeter demo

Demo Board	Processor	PICtail	Comm. Bus
Explorer 16	PIC24FJ128GA010	Ethernet PICtail Plus	SPI
Explorer 16	PIC24FJ128GA010	Fast Ethernet PICtail Plus	SPI
Explorer 16	PIC24FJ128GA010	WiFi PICtail	SPI
Explorer 16	PIC32MX360F512L	Ethernet PICtail Plus	SPI
Explorer 16	PIC32MX360F512L	Fast Ethernet PICtail Plus	SPI
Explorer 16	PIC32MX360F512L	WiFi PICtail	SPI
dsPIC33E USB Starter Kit	33EP512MU810	Fast Ethernet PICtail Plus	SPI2
dsPIC33E USB Starter Kit	33EP512MU810	Fast Ethernet PICtail Plus	PSP Indirect 5
PIC24E USB Starter Kit	24EP512GU810	Fast Ethernet PICtail Plus	SPI2
PIC24E USB Starter Kit	24EP512GU810	Fast Ethernet PICtail Plus	PSP Indirect 5

Google PowerMeter EZConfig demo

Demo Board	Processor	PICtail	Comm. Bus
Explorer 16	PIC24FJ128GA010	WiFi PICtail	SPI
Explorer 16	PIC32MX360F512L	WiFi PICtail	SPI

Energy Monitoring demo

Demo Board	Processor	PICtail	Comm. Bus
Explorer 16	PIC24FJ128GA010	Ethernet PICtail Plus	SPI
Explorer 16	PIC24FJ128GA010	Fast Ethernet PICtail Plus	SPI
Explorer 16	PIC24FJ128GA010	WiFi PICtail	SPI

Software Requirements

These demos will require the [MPLAB IDE](#) (v8.43 or later) and the Pro, Standard, or Lite version of the C30 compiler (v3.21 or later) if using the PIC24F or dsPIC33 architectures, or the Pro, Standard, or Lite version of the C32 compiler (v1.10b or later) if using the PIC32 architecture. If using the PIC24FJ128GA010 with the WiFi PICtail, you will require the Pro or Standard version of the C30 compiler, as the unoptimized code will exceed the part's flash memory size. See Microchip's [compiler page](#) for more information.

Other Tool Requirements

You will need a programmer to program the demo code into the devices. Available options include:

- MPLAB REAL ICE In-Circuit Emulator ([DV244005](#))
- MPLAB ICD 3 In-Circuit Debugger ([DV164035](#))

- PICkit 3 In-Circuit Debugger ([DV164131](#))

3.3 Configuration and Programming

Describes how to configure, compile, and program the demo code into your device.

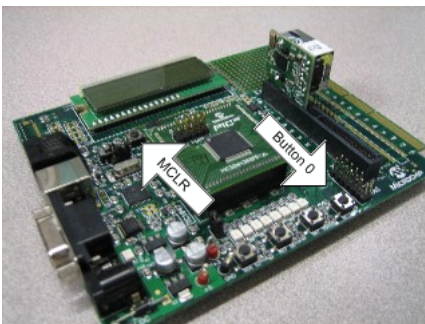
Description

There are several steps to take to configure, compile, and program your code.

1. Open your project directory ("[install directory]\TCPIP\Google PowerMeter", "[install directory]\TCPIP\Google PowerMeter EZConfig", or "[install directory]\TCPIP\Energy Monitoring").
2. Open the workspace that corresponds to your Hardware Setup. For example, the workspace "C30-EX16_PIC24F_MRF24WB.mcw" corresponds to the PIC24FJ128GA010 PIM, using the MRF24WB0M WiFi PICtail. The workspace you open will define a macro that will automatically include the correct configuration files (located in the "Configs" folder).
3. Open the TCPIPConfig.h header file. Search for the MAC address macros (MY_DEFAULT_MAC_BYTEn). Replace the values for MAC bytes 5 and 6 with the hexadecimal representation of the MAC address sticker on your PICtail (e.g. sticker value **12345** would produce MAC address 00:04:A3:00:**30:39**). You can skip this step when using the WiFi or ENC24J600 physical layer (the code will use the built-in MAC addresses if the default value is unchanged).

```
#define MY_DEFAULT_MAC_BYTE1      (0x00)    // Use the default of
#define MY_DEFAULT_MAC_BYTE2      (0x04)    // 00-04-A3-00-00-00 if using
#define MY_DEFAULT_MAC_BYTE3      (0xA3)    // an ENC24J600 or ZeroG ZG2100
#define MY_DEFAULT_MAC_BYTE4      (0x00)    // and wish to use the internal
#define MY_DEFAULT_MAC_BYTE5      (0x00)    // factory programmed MAC
#define MY_DEFAULT_MAC_BYTE6      (0x00)    // address instead.
```

5. If you are using the WiFi PICtail, configure your wireless settings to match your wireless router. See the WiFi Getting Started documentation in the TCP/IP Stack Help File for instructions on how to do this. If you are using the EasyConfig Demo, you can skip this step (you will connect directly to the demo board and configure it via web page).
6. Compile your project and program your device.
7. Press and hold Button 0 on your development board (usually the rightmost or topmost button). Press and release the MCLR button, or remove and reapply power from your board. After four seconds, several LEDs will blink. Release Button 0. This will reset the EEPROM on the development board. You must reset the EEPROM every time you change the device's configuration information (e.g. the Google PowerMeter capture interval or wireless router SSID). This will allow the new configuration information to be uploaded to the EEPROM and prevent out-of-date information from being loaded by the application.



8. Ensure that your board and PC are connected to the same network. If you are using a wired Ethernet solution (ENC28J60 or ENC624J600) plug your computer and board into the same router, or connect them to each other directly with a crossover cable. If you are using the WiFi PICtail Plus, ensure that your computer is connected to the router used for step 5. If you are using the EasyConfig demo, the board will form its own AdHoc network, called "PowerMeterDemo." You should configure your computer to connect directly to this network.
9. Upload the demo web page to the board.
 - Ensure that your computer and your demo board are attached to the same local area network.
 - Open the MPFS2 utility installed with the TCP/IP Stack.

- Set the source directory to the `WebPages` directory in your project directory.
 - Set the output to "BIN Image."
 - Select the project directory in the "Output Files" box.
 - Set the upload path to `http://powermeter` (or to the IP address of your board), using the path `mpfsupload`, the user name `admin`, and the password `microchip`. Note that you can only use the NetBIOS name (`powermeter`) if the board is on the same subnet as the PC you use to connect to it. If you are using the EasyConfig Demo, use the board's IP address (default: 169.254.1.1) instead of its NBNS name.
 - Generate and upload the web page. If the upload is unsuccessful, try using the board's IP address instead of the `http://powermeter` NetBIOS name.
10. If you are using the EasyConfig Demo, open your computer's web browser and navigate to your board's IP address (`http://169.254.1.1` by default). You should be redirected to the Network Configuration page. Click the button to Scan for Wireless Networks. A list of available networks, their signal strength, and their security status will be generated. Click on the network your board will join, and enter any relevant security information. Note that this functionality is unavailable on some browsers (e.g. Internet Explorer 7). Connect your computer to the network you connected your board to.

3.4 Authentication

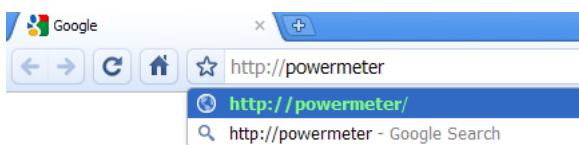
Describes how to authenticate your device.

Description

This section describes how to authenticate your device. Google PowerMeter applications require an authentication step before use. This process creates a data storage area for your meter on the Google server, and provides the device with an authentication token (`auth_token` (see page 38)), authentication path (`auth_path` (see page 37)), and three SHA-1 hash results. These are the hashed values of potential server certificates that Google servers may use. By comparing these hashes to the calculated hash values of a certificate received from a server the device is uploading data to, the device will be able to prevent man-in-the-middle attacks. This compensates for the lack of server certificate validation in Microchip's TCP/IP Stack SSL module.

To authenticate your device:

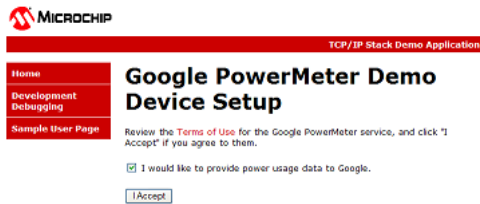
1. Open the web browser on your PC and navigate to `http://powermeter`. Note that if an old IP address association has been cached in your PC's memory for this NetBIOS name, the browser may not be able to navigate to the correct IP address. If this occurs, use the IP address displayed on your board's LCD screen. If you're using a board without an LCD, use Microchip's "TCPIP Discoverer" application (see the TCP/IP Stack help file for more information) to determine your device's IP address.



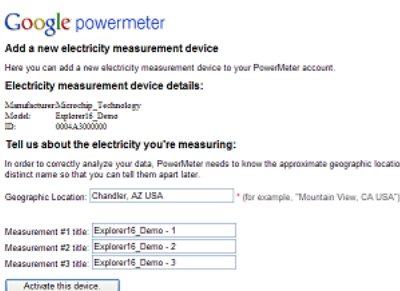
2. The initial index page contains information about the Google PowerMeter project. To authenticate your device, navigate to the "Sample User Page" on the menu.



3. Read Google's Terms of Use and Privacy Policy, and click "I Accept" if you wish to accept the terms outlined within. This page will also give you the option to provide sensor status updates to Google.



4. You will be redirected to a Google-hosted website, and prompted to enter your Gmail address and password. The mechanism that redirects you to this site will upload information about your device, including the device ID, model name, manufacturer name, and the number of sensors. For this demo, the device ID is based off of your board's MAC address and the model and manufacturer are preset.
5. Enter your Gmail address and password, or create a new Gmail account. If you are already signed into Gmail or iGoogle on your PC, you will automatically skip to step 6.
6. You will be redirected to a Google-hosted site that will allow you to add an electricity measurement device. Verify that your device ID is the same as the MAC address you configured. Enter your geographical location, and verify that there are three data source Title boxes. click on "Activate this device" to complete this step.



7. Google will send an HTTP POST message with the activation parameters back to the PIC through the browser. At this point, some browsers may report that the data transmission is not secure; this occurs because the packets sent by the browser to the board are not encrypted. This application model assumes that the device is connected on a secure, trusted local area network, so this lack of encryption should not be a problem. This step won't compromise the activation information to the internet as long as you are using a secure wireless network. If you wish to secure this data on the local area network, you can implement your application so that the board is an SSL server.
8. When the activation data reaches the board, the device will automatically redirect the browser to a URL that will complete and confirm the activation. This page will also allow you to add a Google PowerMeter gadget to your iGoogle home page.

3.5 Operation

Describes the operation of the demos.

Description

The basic demo and EZConfig demo create 3 pseudo-meters.

- The first will generate data on a triangle wave.
- The second will sample the value of the potentiometer and send data proportional to that value.
- During this demo, pressing each push-button on the board will toggle an LED. The third sensor will detect the state of these LEDs and upload a power consumption reading equivalent to one 60W light bulb for every LED on when data is captured.

The Energy Monitoring demo creates a single sensor that corresponds to the power consumption measured by your PIC18F87J72 Energy Monitoring PICtail Plus Daughter Board.

One data measurement for each sensor will be sent to the Google PowerMeter server every 10 minutes- this is the server-limited upload frequency. You can view a graph of these values on your iGoogle page. To view each data point individually, select the "Manage" option on your Google PowerMeter gadget, and choose to "Download Spreadsheet" or "Download Raw Data" for the sensor you are interested in.

By default this demo will attempt to use the Simple Network Time Protocol (SNTP) to determine the values of timestamps of data captures. In rare occasions, some locations may block the UDP port used by the SNTP protocol (port 123). This can prevent the SNTP module from accessing the global SNTP server (pool.ntp.org). If this port is blocked, the code will attempt to establish an HTTP client connection to an external web site and parse the time from the response if the response is an HTTP OK message. If you find that your device is able to complete authentication but does not upload data, it may not be able to get a time stamp value successfully. If this is the case, you can change the SNTP server by redefining the `NTP_SERVER` macro in `SNTP.c` to a local SNTP server or you can change the HTTP server by changing the initialization parameters for the `ServerName` and `RemoteURL` variables in `HTTPTime.c`.

4 Using the Code

Describes how the application works, and how you can use the demo to create a custom application.

Description

Describes how the application works, and how you can use the demo to create a custom application.

4.1 How the Application Works

Describes the operation of the Google PowerMeter application.

Description

These demos are designed to use a cooperative multitasking environment (see the TCP/IP Stack Help file for more information). The Google PowerMeter API is executed as one task in this environment. Every time this task is called, it determines if there are any pending data collection, data transmission, or status transmission tasks to accomplish. The intervals for each of these activities (referred to as preferences) are set by the user in the `InitPowerMeterPreferences` function in `MainDemo.c`. These preferences, along with the TCP/IP Stack's `AppConfig` (see page 37) structure, are stored in EEPROM. Note that the code to store preferences and `AppConfig` information in SPI Flash is not functional yet.

If it is necessary to capture data, the code will determine if each sensor is active, and if so, use the `GetCumulativePower` (see page 18) function to add the number of watt-hours consumed since the last capture to a cumulative power consumption value. It will also add the cumulative value and the time at which it was captured to a queue of data points.

If the task determines that a sufficient interval has passed since the last data transmission attempt, it will open a secure socket to an upload path (determined at activation), and transmit the queued data points. The device functions as an HTTP client for this process.

During these operations, the device stores the count of transmitted packets, count of errors, and other status information in a series of 32-bit counters. When it's time to update device status information, the meter will determine a signal quality value from these counters, and upload this value and other information from these counters to the server. As of May 2010, it is not possible for the user to view the content of these status updates.

4.2 How the Web Page Works

Describes the operation of the Google PowerMeter Demo web page.

Description

Basic Demo

The sample web page included with the Basic Demo application is divided into three parts.

The index page (`index.htm`) contains links to Google PowerMeter information and support pages.

The Development Debugging Page (`debug.htm`) describes and displays the application's preference and configuration information. This will help you debug your application by allowing you to correlate authentication information on your device with the authentication information sent by Google servers (which is viewable on your iGoogle gadget's management page). You can also modify this web page and the `HTTPPrint.h` file to display additional debugging variables (see the TCP/IP Stack Help File's HTTP2 section for information on dynamic variables).

The third page demonstrates the functionality of a page that an end user might see. The link to "Sample User Page" in the menu actually links to the `/terms.htm?` page. The '?' in the link will automatically trigger an HTTP GET operation when the link is clicked. At this point, the device will determine whether it has already been activated.

- If the device has not been activated, it will continue loading the `terms.htm` page. This page will prompt the user to read the terms of use, and give them the checkbox option to automatically transmit status information to the Google servers. Once the user reads the terms and clicks the "I Accept" button, the browser will generate an HTTP POST message containing the status of the checkbox. The device will use the checkbox information to set a flag in the `POWER_METER_PREFERENCES` (see page 32) structure that determines whether status information is sent to the Google servers. It will then construct a Google activation URL and redirect the user's browser to that address. This activation URL will contain a return URL leading to the local `/return.htm` page. The Google servers will send a POST back to `return.htm` (via the browser) containing the activation information. The return page will process this information, then redirect the browser to an activation finalization URL.
- If the device is activated, the HTTP GET processing function will automatically redirect the user's browser to the `/activated.htm` page. This page displays some device parameters using dynamic variables, and contains three form buttons. The first will repeat the activation process. The second will empty the `POWER_METER_PREFERENCES` (see page 32) structure (to indicate that the device is deactivated) and redirect the user to a page that tells them how to manually remove the device from their Google PowerMeter gadget. The third will toggle the status transmission flag.

EZConfig demo

The EZConfig Demo page is similar to the Basic Demo page, with a few key differences. The `index.htm` page will now automatically redirect the browser to `index.htm?`, which will generate an HTTP GET. If the WiFi module is still configured to start in AdHoc mode, the browser will be redirected to `configure.htm`. This page will allow the user to configure the WiFi module to connect to a router, which will act as a gateway to allow the board to upload data to the internet. If the board has already been configured to connect to a gateway, `index.htm` will redirect the user to `info.htm`, which will provide the same information found in the Basic Demo's index page. The user will be able to access the network configuration page through the navigation menu if it's necessary to reconfigure the WiFi module's network connection information.

Energy Monitoring demo

The Energy Monitoring demo uses two pages: the first displays dynamically updated power consumption data from the Energy Monitoring PICtail and the second provides Google PowerMeter activation functionality and configuration information.

4.3 Creating a Custom Application

Describes how to modify the demo code to create a custom application.

Description

There are several steps that must be taken to create a custom application:

- Decide how many sensors your application will use.
 - Change the `GQ_NUM_QUEUES` (see page 36) definition in `gqueue.h` to this number.
 - Change the `GSTATUS_COUNTERS` (see page 30) enumeration typedef in `gcounter.h` to include one `G_SEND_ATTEMPTn` state at the beginning for each sensor (e.g. ensure that the first 4 values in the enumeration are `G_SEND_ATTEMPT0`, `G_SEND_ATTEMPT1`, `G_SEND_ATTEMPT2`, and `G_SEND_ATTEMPT3` if you have 4 sensors). This will allow the module to store the correct number of transmission attempts for each sensor.
 - Change the number of sockets allocated in `TCIPConfig.h`. You will need one `TCP_PURPOSE_DEFAULT` socket for each sensor.
 - Change the `to_send[] SENDER_INFO` array in `gstatus.c` to upload status values for the correct number of sensors. The commented sections in the following code example demonstrate which status parameters must change, depending on the number of sensors. The example shows the proper configuration for three sensors.

```
static SENDER_INFO to_send[] = {
    {(FUNC_OR_STRING) header_get, (FUNC_ARG) 0L, ROM_STRING},
    {(FUNC_OR_STRING) status_url1, (FUNC_ARG) 0L, ROM_STRING},
    {(FUNC_OR_STRING) (ROM_BYTE *)DEVICE_MANUFACTURER, (FUNC_ARG) 0L, ROM_STRING},
    {(FUNC_OR_STRING) status_url2, (FUNC_ARG) 0L, ROM_STRING},

```



```

{ (FUNC_OR_STRING) (ROM BYTE *)DEVICE_MODEL, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) device_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) DeviceId, (FUNC_ARG) 0L, FUNC_CALL},
{ (FUNC_OR_STRING) build_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) DeviceBuild, (FUNC_ARG) 0L, FUNC_CALL},
{ (FUNC_OR_STRING) uptime_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) Uptime, (FUNC_ARG) 0L, FUNC_CALL},
{ (FUNC_OR_STRING) MaybeReboot, (FUNC_ARG) 0L, FUNC_CALL},
{ (FUNC_OR_STRING) SignalQuality, (FUNC_ARG) 0L, FUNC_CALL}, // Get signal quality for
sensor 1 (offset 0)
{ (FUNC_OR_STRING) SignalQuality, (FUNC_ARG) 1L, FUNC_CALL}, // Get signal quality for
sensor 2 (offset 1)
{ (FUNC_OR_STRING) SignalQuality, (FUNC_ARG) 2L, FUNC_CALL}, // Get signal quality for
sensor 3 (offset 2)
{ (FUNC_OR_STRING) xfer_attempt_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_SEND_ATTEMPT0, FUNC_CALL},
{ (FUNC_OR_STRING) xfer_success_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_SEND_SUCCESS, FUNC_CALL},
{ (FUNC_OR_STRING) xfer_time_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) XferTime, (FUNC_ARG) 0L, FUNC_CALL},
{ (FUNC_OR_STRING) xfer_result_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_SEND_RESULT, FUNC_CALL},
{ (FUNC_OR_STRING) capture_attempt_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_CAPTURE_ATTEMPT,
FUNC_CALL},
{ (FUNC_OR_STRING) capture_success_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_CAPTURE_SUCCESS,
FUNC_CALL},
{ (FUNC_OR_STRING) error_bad_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_ERROR_BAD_STATE,
FUNC_CALL},
{ (FUNC_OR_STRING) error_timeout_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_ERROR_TIMEOUT, FUNC_CALL},
{ (FUNC_OR_STRING) error_space_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_ERROR_NO_SEND_SPACE,
FUNC_CALL},
{ (FUNC_OR_STRING) error_value_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_ERROR_BAD_VALUE,
FUNC_CALL},
{ (FUNC_OR_STRING) error_socket_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_ERROR_NO_SOCKET,
FUNC_CALL},
{ (FUNC_OR_STRING) error_status_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_STATUS_RESULT, FUNC_CALL},
{ (FUNC_OR_STRING) error_batch_success_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_BATCH_SEND_SUCCESS,
FUNC_CALL},
{ (FUNC_OR_STRING) error_batch_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) GenCounter, (FUNC_ARG) (INT32) G_BATCH_SEND_RESULT,
FUNC_CALL},
{ (FUNC_OR_STRING) QueueSize, (FUNC_ARG) 0L, FUNC_CALL}, // Get queue size for sensor 1
(offset 0)
{ (FUNC_OR_STRING) QueueSize, (FUNC_ARG) 1L, FUNC_CALL}, // Get queue size for sensor 2
(offset 1)
{ (FUNC_OR_STRING) QueueSize, (FUNC_ARG) 2L, FUNC_CALL}, // Get queue size for sensor 3
(offset 2)
{ (FUNC_OR_STRING) PktCount, (FUNC_ARG) 0L, FUNC_CALL}, // Get packet TX count for
sensor 1 (offset 0)
{ (FUNC_OR_STRING) PktCount, (FUNC_ARG) 1L, FUNC_CALL}, // Get packet TX count for
sensor 2 (offset 1)
{ (FUNC_OR_STRING) PktCount, (FUNC_ARG) 2L, FUNC_CALL}, // Get packet TX count for
sensor 3 (offset 2)
{ (FUNC_OR_STRING) PktSkip, (FUNC_ARG) 0L, FUNC_CALL}, // Get packet skip count for
sensor 1 (offset 0)
{ (FUNC_OR_STRING) PktSkip, (FUNC_ARG) 1L, FUNC_CALL}, // Get packet skip count for
sensor 2 (offset 1)
{ (FUNC_OR_STRING) PktSkip, (FUNC_ARG) 2L, FUNC_CALL}, // Get packet skip count for
sensor 3 (offset 2)
{ (FUNC_OR_STRING) pub_hash_success_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) PubHashSuccess, (FUNC_ARG) 0L, FUNC_CALL},
{ (FUNC_OR_STRING) pub_hash_attempt_param, (FUNC_ARG) 0L, ROM_STRING},
{ (FUNC_OR_STRING) PubHashAttempt, (FUNC_ARG) 0L, FUNC_CALL},

```



```
{(FUNC_OR_STRING) header_http_host, (FUNC_ARG) 0L, ROM_STRING},
{(FUNC_OR_STRING) goog_host, (FUNC_ARG) 0L, ROM_STRING},
{(FUNC_OR_STRING) header_agent_connection, (FUNC_ARG) 0L, ROM_STRING},
{(FUNC_OR_STRING) ((ROM BYTE *) NULL), (FUNC_ARG) 0L, NO_MORE},
};
```

- Change the implementation of the `GetCumulativePower` (see page 18) function in the `ginsu.c` file. This function is called when the code attempts to capture data. It accepts three arguments: `which` indicates which sensor or meter the code is accessing, `timestamp` points to a location to which the UTC timestamp should be written, and `watts` points to a location to which a 64-bit cumulative sum of instantaneous power readings (in watt-hours) should be written. You can use the `GetPreferenceInt32` (see page 20) function to determine the capture interval; this is the time period since the last capture. You can use this and an instantaneous measure of power consumption to estimate the amount of power consumed since the last capture. See the LED-based example meter for a demonstration. Note that as the capture interval decreases, the accuracy of the estimate increases. Decreasing the capture interval too far can increase the risk of overrunning the end of the queue and losing data, though.
- Change the other functions that use the 'which' parameter to reflect the number of sensors that you'd like to use. This includes the functions `GetSQ` (see page 21), `GetSQDetails` (see page 22), and `SensorStillActive` (see page 26) in `ginsu.c`.
- Review the function implementation in `ginsu.c`, and customize any functions that require different implementation for your application.
- Change the manufacturer name and model number of your device. These values are defined as `DEVICE_MANUFACTURER` (see page 34) and `DEVICE_MODEL` (see page 34) in `ginsu.h`. You can also change the method for generating device IDs (in `CustomHTTPApp.c` and `gstatus.c`); using the MAC address as the device ID will ensure that each device you produce has a unique ID, though. The `DeviceID` function in `gstatus.c` provides the device ID for status messages, and the `HTTPPostAuthenticate` (see page 15) and `HTTPPostAuthInfo` (see page 15) functions in `CustomHTTPApp.c` provide it for authentication.
- Change the value of the `HTTP_MAX_DATA_LEN` macro in `TCPIPConfig.h`. This macro controls the size of the buffer used for sending or receiving POST data. You may need to increase this value depending on the length of the `DEVICE_MANUFACTURER` (see page 34) and `DEVICE_MODEL` (see page 34) macros, or if there is a change in the Google PowerMeter API that would increase the length of transmitted or received POST messages.
- Customize your activation web page. You will also need to determine how the device attaches to your local network to be activated. For one device, the NetBIOS name can provide access to the device, but for multiple devices on one network, some method must be implemented to determine which device is being accessed.

5 Application API




Describes functions and structures used by this demo.

5.1 Functions

Describes the functions used by this demo.

5.1.1 CustomHTTPApp

Functions

	Name	Description
	HTTPDeactivateDevice (↗ see page 14)	Deactivates a device.
	HTTPPostAuthenticate (↗ see page 15)	Redirects a browser to the activation URL on Google's site.
	HTTPPostAuthInfo (↗ see page 15)	Parses data from a post-activation POST message.

Description

Describes web-page handling functions implemented in CustomHTTPApp.c.

5.1.1.1 HTTPDeactivateDevice Function

Deactivates a device.

File

CustomHTTPApp.c

C

```
static HTTP_IO_RESULT HTTPDeactivateDevice();
```

Description

This function will invalidate the token value in the Google PowerMeter preferences structure. This will prevent the device from uploading data. This function will not remove the device from the Google PowerMeter gadget on the user's iGoogle page.

Preconditions

None

Return Values

Return Values	Description
HTTP_IO_DONE	the parameter has been found and saved

Function

```
static HTTP_IO_RESULT HTTPDeactivateDevice(void)
```

5.1.1.2 HTTPPostAuthenticate Function

Redirects a browser to the activation URL on Google's site.

File

CustomHTTPApp.c

C

```
static HTTP_IO_RESULT HTTPPostAuthenticate();
```

Description

This function will load the activation URL into the curHTTP.data buffer and then queue up an HTTP redirect action for the browser that caused this function to be called. This will cause the activation information for the device to be submitted to Google servers.

Preconditions

None

Return Values

Return Values	Description
HTTP_IO_DONE	the parameter has been found and saved
HTTP_IO_WAITING	the function is pausing to continue later
HTTP_IO_NEED_DATA	data needed by this function has not yet arrived

Function

```
static HTTP_IO_RESULT HTTPPostAuthenticate(void)
```

5.1.1.3 HTTPPostAuthInfo Function

Parses data from a post-activation POST message.

File

CustomHTTPApp.c

C

```
static HTTP_IO_RESULT HTTPPostAuthInfo();
```

Description

After activation, the Google servers will redirect the browser back to the return URL provided to them with a message containing POST data. This function will parse the data to check the security nonce, read the authentication token and path, and read the server certificate hashes. If these values were received successfully, the function will load the URL that completes the activation process into curHTTP.data and then redirect the browser to that URL.

Preconditions

None

Return Values

Return Values	Description
HTTP_IO_DONE	the parameter has been found and saved
HTTP_IO_WAITING	the function is pausing to continue later
HTTP_IO_NEED_DATA	data needed by this function has not yet arrived

Function

```
static HTTP_IO_RESULT HTTPPostAuthInfo(void)
```

5.1.2 ginsu

Describes Google PowerMeter interface functions implemented in ginsu.c.

Functions

	Name	Description
⇒	CalculatePubKeyHash (↗ see page 17)	Calculate the public key hash for the current HTTPS socket.
⇒	ConvertUtcSecToString (↗ see page 17)	Converts a UTC timestamp to a string
⇒	GetBuildNumber (↗ see page 18)	Returns the build number
⇒	GetCumulativePower (↗ see page 18)	Gets the current cumulative power for a given MTU.
⇒	GetHashListEntry (↗ see page 19)	Returns a pointer to a public key hash.
⇒	GetIpv4 (↗ see page 19)	Returns the device's IPv4 address
⇒	GetMac (↗ see page 20)	Returns the device's MAC address
⇒	GetPreferenceInt32 (↗ see page 20)	Returns an INT32-type power meter preference
⇒	GetPreferenceStr (↗ see page 21)	Returns a string-type power meter preference
⇒	GetSQ (↗ see page 21)	Returns a signal quality value for the MTU.
⇒	GetSQDetails (↗ see page 22)	Returns details about the signal quality value.
⇒	GetUptimeSec (↗ see page 22)	Returns the time that the stack has been active
⇒	GetUtcTimeSec (↗ see page 23)	Returns a UTC timestamp
⇒	GetWebPort (↗ see page 23)	Returns the web port to use to communicate
⇒	GInitForWork (↗ see page 24)	Initializes the Google PowerMeter state machine.
⇒	GPollForWork (↗ see page 24)	Task for the Google PowerMeter application
⇒	GSendInit (↗ see page 24)	Sets the next communication to an initial communication.
⇒	GUpdateNVRam (↗ see page 25)	Updates non-volatile memory with the current hash values
⇒	PadBuffer (↗ see page 25)	Pads a hour/minute/second value to include a leading zero
⇒	SensorStillActive (↗ see page 26)	Determines if a given sensor is still active.
⇒	SetHashListEntry (↗ see page 26)	Sets the value of a public key hash
⇒	SetupCalculatePubKeyHash (↗ see page 27)	Sets up the check of the public key hash from the Google servers.

Description

Describes Google PowerMeter interface functions implemented in ginsu.c.

5.1.2.1 CalculatePubKeyHash Function

Calculate the public key hash for the current HTTPS socket.

File

ginsu.h

C

```
void CalculatePubKeyHash(  
    UINT8 socket,  
    INT8 * current_hash  
);
```

Description

Calculates the public key hash for the specified socket. When the HTTPS socket was opened, its public key information was copied into the local G_PUBKEY_INFO structure. This function will hash that data.

Preconditions

None

Parameters

Parameters	Description
socket	The socket that will be hashed.
current_hash	Pointer to the return value for the hash calculation.

Function

```
void CalculatePubKeyHash(UINT8 socket, INT8 *current_hash)
```

5.1.2.2 ConvertUtcSecToString Function

Converts a UTC timestamp to a string

File

ginsu.h

C

```
void ConvertUtcSecToString(  
    INT32 utc,  
    char* time_str  
);
```

Description

This function will convert a UTC timestamp to an RFC-specified string format representation of that value.

Preconditions

None

Parameters

Parameters	Description
utc	The UTC timestamp to be converted
time_str	A string pointing to the buffer that will contain the result.

Function

```
void ConvertUtcSecToString(INT32 utc, char* time_str)
```

5.1.2.3 GetBuildNumber Function

Returns the build number

File

ginsu.h

C

```
INT32 GetBuildNumber( );
```

Description

Returns the build number

Preconditions

None

Return Values

Return Values	Description
INT32	The build number for this demo.

Function

```
INT32 GetBuildNumber(void)
```

5.1.2.4 GetCumulativePower Function

Gets the current cumulative power for a given MTU.

File

ginsu.h

C

```
void GetCumulativePower(  
    INT16 which,  
    INT32 * timestamp,  
    INT64 * watts  
);
```

Description

This function will obtain a reading of the current cumulative power consumed by the MTU, in watt-hours. This function will sample the 'meter' sources, correct the readings based on the capture interval, and add them to the cumulative reading.

Preconditions

None

Parameters

Parameters	Description
which	Indicates which MTU to return the power value for
timestamp	Pointer to the INT32 into which the current UTC time should be copied
watts	Pointer to the INT64 into which the current cumulative power should be copied

Function

```
void GetCumulativePower(INT15 which, INT32 *timestamp, INT64 *watts)
```

5.1.2.5 GetHashListEntry Function

Returns a pointer to a public key hash.

File

ginsu.h

C

```
const INT8 * GetHashListEntry(
    INT8 hash_slot
);
```

Description

Returns a pointer to a public key hash.

Preconditions

None

Parameters

Parameters	Description
hash_slot	Specifies the hash to locate.

Return Values

Return Values	Description
const INT8 *	The hash value pointer

Function

```
const INT8 *GetHashListEntry(INT8 hash_slot)
```

5.1.2.6 GetIpv4 Function

Returns the device's IPv4 address

File

ginsu.h

C

```
UINT32 GetIpv4();
```

Description

Returns the device's IPv4 address

Preconditions

None

Return Values

Return Values	Description
UINT32	The device's IPv4 address

Function

```
UINT32 GetIpv4(void)
```

5.1.2.7 GetMac Function

Returns the device's MAC address

File

ginsu.h

C

```
const UINT8 * GetMac( );
```

Description

Returns the device's MAC address. Byte[0] is the most significant byte; byte[5] is the least significant byte. This value is not null-terminated.

Preconditions

None

Return Values

Return Values	Description
UINT8*	Pointer to the MAC address

Function

```
const UINT8 *GetMac(void)
```

5.1.2.8 GetPreferenceInt32 Function

Returns an INT32-type power meter preference

File

ginsu.h

C

```
INT32 GetPreferenceInt32(  
    PREF_LIST which  
);
```

Description

This function returns an INT32-type power meter preference.

Preconditions

None

Parameters

Parameters	Description
which	Specified which preference to return. <ul style="list-style-type: none">CAPTURE_SEC_INT - The capture interval frequency (seconds)SEND_SEC_INT - The data transmission frequency (seconds)SEND_STATUS_INT - The status transmission frequency (seconds)

Return Values

Return Values	Description
INT32	The preference value.

Function

INT32 GetPreferenceInt32(PREF_LIST (see page 33) which)

5.1.2.9 GetPreferenceStr Function

Returns a string-type power meter preference

File

ginsu.h

C

```
const char * GetPreferenceStr(
    PREF_LIST which
);
```

Description

This function returns a string-type power meter preference.

Preconditions

None

Parameters

Parameters	Description
which	The preference to return <ul style="list-style-type: none">AUTH_TOKEN_STR - The authorization tokenAUTH_PATH_STR - The authorization path

Return Values

Return Values	Description
const char *	Pointer to the preference string.

Function

const char *GetPreferenceStr(PREF_LIST (see page 33) which)

5.1.2.10 GetSQ Function

Returns a signal quality value for the MTU.

File

ginsu.h

C

```
UINT32 GetSQ(
    INT16 which
);
```

Description

Returns a signal quality value for the MTU. This function returns non- zero if the sensor is active. If zero is returned, then assume there is no sensor. If the packet error count is high, it may be possible to get some packets but end up with a zero signal quality value. In this case, the sensor can be treated as missing.

Preconditions

None

Parameters

Parameters	Description
which	Indicates which MTU quality to check.

Return Values

Return Values	Description
UINT32	The signal quality value.

Function

UINT32 GetSQ(INT16 which)

5.1.2.11 GetSQDetails Function

Returns details about the signal quality value.

File

ginsu.h

C

```
void GetSQDetails(  
    INT16 which,  
    UINT32 * count,  
    UINT32 * skipped  
);
```

Description

This function returns the count of transmission attempts and the number of problems that have occurred.

Preconditions

None

Parameters

Parameters	Description
which	Indicates which MTU to check.
count	Return value for the count of packet transmission attempts.
skipped	Return value for the number of packet transmission failures.

Function

void GetSQDetails(INT16 which, UINT32 *count, UINT32 *skipped)

5.1.2.12 GetUptimeSec Function

Returns the time that the stack has been active

File

ginsu.h

C

```
INT32 GetUptimeSec();
```

Description

This function returns that amount of time that has passed since the stack began running.

Preconditions

None

Return Values

Return Values	Description
INT32	The number of seconds the stack has been running.

Function

INT32 GetUptimeSec(void)

5.1.2.13 GetUtcTimeSec Function

Returns a UTC timestamp

File

ginsu.h

C

```
INT32 GetUtcTimeSec();
```

Description

This function returns a UTC timestamp to the Google PowerMeter code.

Preconditions

None

Return Values

Return Values	Description
INT32	The UTC timestamp

Function

INT32 GetUtcTimeSec(void)

5.1.2.14 GetWebPort Function

Returns the web port to use to communicate

File

ginsu.h

C

```
UINT16 GetWebPort();
```

Description

Returns the web port to use to communicate

Preconditions

None

Return Values

Return Values	Description
UINT16	The web port

Function

UINT16 GetWebPort(void)

5.1.2.15 GInitForWork Function

Initializes the Google PowerMeter state machine.

File

ginsu.h

C

```
void GInitForWork();
```

Description

This code will initialize the Google PowerMeter state machine. It will set the initial capture attempt timestamp and set the flag indicating that the next transmission will be an initial transmission. This function should be called while the system is booting after the device modules (like the TCP/IP stack and sensors) are initialized.

Preconditions

None

Function

```
void GInitForWork(void)
```

5.1.2.16 GPollForWork Function

Task for the Google PowerMeter application

File

ginsu.h

C

```
void GPollForWork();
```

Description

This function checks if any Google PowerMeter subtasks must be executed. If so, it will execute them. This function must be called periodically to enable the Google PowerMeter functionality.

Preconditions

None

Function

```
void GPollForWork(void)
```

5.1.2.17 GSendInit Function

Sets the next communication to an initial communication.

File

ginsu.h

C

```
void GSendInit(  
    UINT8 send_now  
);
```

Description

This function forces GSend() to treat the next communication as an initial communication.

Preconditions

None

Parameters

Parameters	Description
send_now	Forces the module to send any queued data immediately.

Function

```
void GSendInit(UINT8 send_now)
```

5.1.2.18 GUpdateNVRam Function

Updates non-volatile memory with the current hash values

File

ginsu.h

C

```
void GUpdateNVRam( ) ;
```

Description

Updates non-volatile memory with the current hash values

Preconditions

None

Function

```
void GUpdateNVRam(void)
```

5.1.2.19 PadBuffer Function

Pads a hour/minute/second value to include a leading zero

File

ginsu.c

C

```
void PadBuffer(  
    DWORD value,  
    BYTE * buffer  
) ;
```

Description

This function will check a time value to determine if it's a single digit value. If so, it will insert a leading zero.

Preconditions

None

Parameters

Parameters	Description
value	The time value
buffer	The string to be updated

Function

void PadBuffer (DWORD value, BYTE * buffer)

5.1.2.20 SensorStillActive Function

Determines if a given sensor is still active.

File

ginsu.h

C

```
INT8 SensorStillActive(
    INT16 which_sensor
);
```

Description

Determines if a given sensor is still active.

Preconditions

None

Parameters

Parameters	Description
which_sensor	The sensor to check

Return Values

Return Values	Description
TRUE	The sensor is active
FALSE	The sensor is inactive

Function

INT8 SensorStillActive(INT16 which_sensor)

5.1.2.21 SetHashListEntry Function

Sets the value of a public key hash

File

ginsu.h

C

```
void SetHashListEntry(
    INT8 * hash,
    INT8 hash_slot
);
```

Description

Sets the value of a public key hash

Preconditions

None

Parameters

Parameters	Description
hash	Pointer to the new hash value.

hash_slot	The hash to replace with the new value.
-----------	---

Function

```
void SetHashListEntry(INT8 *hash, INT8 hash_slot)
```

5.1.2.22 SetupCalculatePubKeyHash Function

Sets up the check of the public key hash from the Google servers.

File

ginsu.h

C

```
void SetupCalculatePubKeyHash(  
    UINT8 socket  
);
```

Description

Sets up the check of the public key hash from the Google servers. This function will clear the G_PUBKEY_INFO structure.

Preconditions

None

Parameters





Parameters	Description
socket	The socket to set up the hash for.

Function

```
void SetupCalculatePubKeyHash(UINT8 socket)
```

5.1.3 MainDemo

Functions

	Name	Description
	InitNVMemContents (see page 27)	None
	ProcessIO (see page 28)	Processes A/D data from the potentiometer
	SaveNVMemContents (see page 28)	Writes configuration values to non-volatile memory
	SavePowerMeterPreferences (see page 29)	Writes Power Meter preferences to non-volatile storage

Description

Describes application functions implemented in MainDemo.c.

5.1.3.1 InitNVMemContents Function

File

MainDemo.c

C

```
static void InitNVMemContents();
```

Side Effects

None

Returns

Write/Read non-volatile config variables.

Description

None

Remarks

None

Preconditions

MPFSInit() is already called.

Function

```
void InitNVMemContents(void)
```

5.1.3.2 ProcessIO Function

File

MainDemo.c

C

```
static void ProcessIO();
```

Description

Processes A/D data from the potentiometer

5.1.3.3 SaveNVMemContents Function

File

MainDemo.h

C

```
void SaveNVMemContents();
```

Side Effects

None

Returns

None.

Description

Writes configuration values to non-volatile memory

Remarks

None

Preconditions

Valid config values have been loaded or generated

Function

void SaveNVMemContents(void)

5.1.3.4 SavePowerMeterPreferences Function

File

MainDemo.h

C

```
void SavePowerMeterPreferences ( ) ;
```

Side Effects

None

Returns

None.

Description

Writes Power Meter preferences to non-volatile storage

Remarks

None

Preconditions








Valid Power Meter preferences have been loaded or generated

Function



void SavePowerMeterPreferences(void)



5.2 Structures

Enumerations

	Name	Description
	GSTATUS_COUNTERS (↗ see page 30)	Enumeration of application counters
	gstatus_counters_type (↗ see page 30)	Enumeration of application counters
	NV_MEM_STRUCTURE_OFFSETS (↗ see page 31)	Enumeration for non-volatile memory allocation
	PMButtonState (↗ see page 32)	Define an enumeration to handle our button-press state machine
	PMCaptureMode (↗ see page 32)	Defines an enumeration that corresponds to our sensors
	PREF_LIST (↗ see page 33)	Enumeration for device preferences
	_pref_list (↗ see page 33)	Enumeration for device preferences

Structures

	Name	Description
	G_PKEY_INFO (↗ see page 30)	Structure to store public key information
	_g_pkey_info (↗ see page 30)	Structure to store public key information

	POWER_METER_PREFERENCES ( see page 32)	Structure defining Google PowerMeter Preferences (configuration options)
---	---	--

Description

Describes the structures used by this demo.

5.2.1 G_PKEY_INFO Structure

File

ginsu.h

C

```
typedef struct _g_pkey_info {
    INT16 pub_size_bytes;
    INT8 pub_key[128];
    INT8 pub_e[3];
    UINT8 pub_guid;
} G_PKEY_INFO;
```

Members

Members	Description
INT16 pub_size_bytes;	The size of the public key, in bytes (128 max)
INT8 pub_key[128];	The public key modulus
INT8 pub_e[3];	The public key exponent
UINT8 pub_guid;	This is used as a TCP_SOCKET which is a BYTE

Description

Structure to store public key information

5.2.2 GSTATUS_COUNTERS Enumeration

Enumeration of application counters

File

gcounter.h

C

```
typedef enum gstatus_counters_type{
    G_SEND_ATTEMPT0,
    G_SEND_ATTEMPT1,
    G_SEND_ATTEMPT2,
    G_SEND_SUCCESS,
    G_SEND_RESULT,
    G_BATCH_SEND_SUCCESS,
    G_BATCH_SEND_RESULT,
    G_STATUS_ATTEMPT,
    G_STATUS_SUCCESS,
    G_STATUS_RESULT,
    G_PUBKEY_ATTEMPT,
    G_PUBKEY_SUCCESS,
    G_ERROR_TIMEOUT,
    G_ERROR_BAD_STATE,
    G_ERROR_NO_SEND_SPACE,
    G_ERROR_BAD_VALUE,
    G_ERROR_NO_SOCKET,
    G_CAPTURE_ATTEMPT,
```

```

    G_CAPTURE_SUCCESS,
    G_TOTAL_STATUS_COUNTERS
} GSTATUS_COUNTERS;

```

Members

Members	Description
G_SEND_ATTEMPT0	Send attempts for sensor 0
G_SEND_ATTEMPT1	Send attempts for sensor 1
G_SEND_ATTEMPT2	Send attempts for sensor 2
G_SEND_SUCCESS	Count of successful data transmissions
G_SEND_RESULT	Last data TX HTTP response code
G_BATCH_SEND_SUCCESS	Batch TX success count
G_BATCH_SEND_RESULT	Batch TX HTTP response code
G_STATUS_ATTEMPT	Send attempts for status information
G_STATUS_SUCCESS	Count of successful status transmissions
G_STATUS_RESULT	Last status TX HTTP response code
G_PUBKEY_ATTEMPT	Public key attempts
G_PUBKEY_SUCCESS	Public key successes
G_ERROR_TIMEOUT	Number of timeout errors
G_ERROR_BAD_STATE	Number of bad state errors
G_ERROR_NO_SEND_SPACE	Number of errors caused by insufficient socket space
G_ERROR_BAD_VALUE	Number of bad value errors
G_ERROR_NO_SOCKET	Number of socket allocation failure errors
G_CAPTURE_ATTEMPT	Number of capture attempts
G_CAPTURE_SUCCESS	Number of capture successes
G_TOTAL_STATUS_COUNTERS	Must always be the last enum value

Description

These are the various events we want to keep track of. It is assumed there is a 4 byte entity behind each counter. In theory the counter can also be used to hold a value instead (such as a failure code) by using get/set instead of increment.

5.2.3 NV_MEM_STRUCTURE_OFFSETS Enumeration

File

MainDemo.h

C

```

typedef enum {
    identifierOffset = 0,
    appConfigOffset = 1,
    powerMeterPreferencesOffset = sizeof(APP_CONFIG)+1,
    structureEndOffset = powerMeterPreferencesOffset+sizeof(POWER_METER_PREFERENCES)
} NV_MEM_STRUCTURE_OFFSETS;

```

Description

Enumeration for non-volatile memory allocation

5.2.4 PMButtonState Enumeration

File

MainDemo.h

C

```
typedef enum {  
    SM_IDLE = 0u,  
    SM_DEBOUNCE_DOWN,  
    SM_RELEASE_WAIT,  
    SM_DEBOUNCE_UP  
} PMButtonState;
```

Description

Define an enumeration to handle our button-press state machine

5.2.5 PMCaptureMode Enumeration

File

MainDemo.h

C

```
typedef enum {  
    CM_TRIANGLE = 0u,  
    CM_POT,  
    CM_LEDS  
} PMCaptureMode;
```

Members

Members	Description
CM_TRIANGLE = 0u	Triangle wave sample sensor
CM_POT	Potentiometer-based sample sensor
CM_LEDS	LED-based sample sensor

Description

Defines an enumeration that corresponds to our sensors

5.2.6 POWER_METER_PREFERENCES Structure

File

MainDemo.h

C

```
typedef struct {  
    BYTE auth_token[65];  
    BYTE auth_path[193];  
    DWORD snonce;  
    BYTE pKeyHashes[3][20];  
    WORD cap_sec_interval;  
    WORD send_sec_interval;  
    BOOL send_status;
```

```
} POWER_METER_PREFERENCES;
```

Members

Members	Description
BYTE auth_token[65];	The user identification value
BYTE auth_path[193];	The data upload path
DWORD snonce;	A security nonce
BYTE pKeyHashes[3][20];	SHA-1 hashes of potential Google SSL certificates
WORD cap_sec_interval;	Data capture interval (in seconds)
WORD send_sec_interval;	Data transmission interval (in seconds)
BOOL send_status;	Boolean - indicates whether to send status information

Description

Structure defining Google PowerMeter Preferences (configuration options)

5.2.7 PREF_LIST Enumeration

File

ginsu.h

C

```
typedef enum _pref_list {
    AUTH_TOKEN_STR,
    AUTH_PATH_STR,
    CAPTURE_SEC_INT,
    SEND_SEC_INT,
    SEND_STATUS_INT
} PREF_LIST;
```

Members







Members	Description
AUTH_TOKEN_STR	Authorization token
AUTH_PATH_STR	Authorization path
CAPTURE_SEC_INT	Frequency of capturing data (seconds)
SEND_SEC_INT	Frequency of sending data to google (seconds)
SEND_STATUS_INT	boolean - send status to google (true) or not.
















Description

Enumeration for device preferences

5.3 Macros

Macros

	Name	Description
	APPLICATION_BUILD ( see page 34)	Defines the build number
	DEVICE_MANUFACTURER ( see page 34)	Defines the device manufacturer
	DEVICE_MODEL ( see page 34)	Defines the device model

	DEVICE_NUM_SENSORS ( see page 35)	Defines the number of sensors the device is using This value is based on the GQ_NUM_QUEUES ( see page 36) macro, defined in gqueue.h
	GOOGLE_DATA_PORT ( see page 35)	Data port (https)
	GOOGLE_HOST ( see page 35)	Google host address
	GOOGLE_STATUS_PORT ( see page 35)	Status port (https)
	GQ_NUM_QUEUES ( see page 36)	The number of queues we can support
	MAX_NUM_PUBKEY_HASH ( see page 36)	Defines the maximum number of hashes stored by the application
	PUBKEY_HASH_SIZE ( see page 36)	Defines the size of a public key hash in bytes (SHA-1)

Description

Describes the macros used by this demo.

5.3.1 APPLICATION_BUILD Macro

File

MainDemo.h

C

```
#define APPLICATION_BUILD 0x62
```

Description

Defines the build number

5.3.2 DEVICE_MANUFACTURER Macro

File

MainDemo.h

C

```
#define DEVICE_MANUFACTURER "Microchip_Technology"
```

Description

Defines the device manufacturer

5.3.3 DEVICE_MODEL Macro

File

MainDemo.h

C

```
#define DEVICE_MODEL "Explorer16_Demo"
```

Description

Defines the device model

5.3.4 DEVICE_NUM_SENSORS Macro

File

MainDemo.h

C

```
#define DEVICE_NUM_SENSORS GQ_NUM_QUEUES
```

Description

Defines the number of sensors the device is using This value is based on the GQ_NUM_QUEUES (see page 36) macro, defined in gqueue.h

5.3.5 GOOGLE_DATA_PORT Macro

File

ginsu.h

C

```
#define GOOGLE_DATA_PORT 443 // Data port (https)
```

Description

Data port (https)

5.3.6 GOOGLE_HOST Macro

File

ginsu.h

C

```
#define GOOGLE_HOST "www.google.com" // Google host address
```

Description

Google host address

5.3.7 GOOGLE_STATUS_PORT Macro

File

ginsu.h

C

```
#define GOOGLE_STATUS_PORT 443 // Status port (https)
```

Description

Status port ([https](#))

5.3.8 GQ_NUM_QUEUES Macro

File

gqueue.h

C

```
#define GQ_NUM_QUEUES 3
```

Description

The number of queues we can support

5.3.9 MAX_NUM_PUBKEY_HASH Macro

File

ginsu.h

C

```
#define MAX_NUM_PUBKEY_HASH 3
```

Description

Defines the maximum number of hashes stored by the application

5.3.10 PUBKEY_HASH_SIZE Macro

File

ginsu.h

C





```
#define PUBKEY_HASH_SIZE 20
```













Description

Defines the size of a public key hash in bytes (SHA-1)

5.4 Variables

Variables

	Name	Description
	AppConfig (↗ see page 37)	Declare AppConfig structure
	auth_path (↗ see page 37)	Pointer to the authorization path
	auth_token (↗ see page 38)	Pointer to the authorization token
	captureMode (↗ see page 38)	Declare state tracking variables for buttons pushes and capture modes

	gActivationConfirm (see page 38)	First fragment of the activation confirmation URL. Use gActivationString2 (see page 39) and gActivationString3 (see page 39) as the second and third fragments.
	gActivationString1 (see page 38)	First fragment of the activation URL. Continues until manufacturer name is required.
	gActivationString2 (see page 39)	Second fragment of the activation URL. Continues until model name is required.
	gActivationString3 (see page 39)	Third fragment of the activation URL. Continues until device ID is required.
	gActivationString4 (see page 39)	Fourth fragment of the activation URL. Continues until the number of cumulative variables is required.
	gActivationString5 (see page 39)	Fifth fragment of the activation URL. Continues until the return URL is required.
	gActivationString6 (see page 40)	Sixth fragment of the activation URL. Continues until the security nonce is required. Contains the end of the return URL.
	gCumulativePower (see page 40)	Declare an INT64 to contain the cumulative power consumption of the device or circuit you're monitoring
	goog_host (see page 40)	Definition of the Google host name; used in data and status messages
	gpkey_info (see page 40)	Public Key Structure Information
	gPowerMeterPreferences (see page 41)	Declare POWER_METER_PREFERENCES (see page 32) structure to contain required settings
	StackStartTime (see page 41)	Declare a variable to hold the stack's relative start time

Description

Describes the variables used by this demo.

5.4.1 AppConfig Variable

File

MainDemo.c

C

```
APP_CONFIG AppConfig;
```

Description

Declare AppConfig structure

5.4.2 auth_path Variable

File

ginsu.c

C

```
const char * auth_path = (const char *)gPowerMeterPreferences.auth_path;
```

Description

Pointer to the authorization path

5.4.3 auth_token Variable

File

ginsu.c

C

```
const char * auth_token = (const char *)gPowerMeterPreferences.auth_token;
```

Description

Pointer to the authorization token

5.4.4 captureMode Variable

File

MainDemo.h

C

```
PMCaptureMode captureMode;
```

Description

Declare state tracking variables for buttons pushes and capture modes

5.4.5 gActivationConfirm Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationConfirm[] = "https://www.google.com/powermeter/device/finish?mfg=";
```

Description

First fragment of the activation confirmation URL. Use gActivationString2 (see page 39) and gActivationString3 (see page 39) as the second and third fragments.

5.4.6 gActivationString1 Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationString1[] = "https://www.google.com/powermeter/device/activate?mfg=";
```

Description

First fragment of the activation URL. Continues until manufacturer name is required.

5.4.7 gActivationString2 Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationString2[] = "&model=";
```

Description

Second fragment of the activation URL. Continues until model name is required.

5.4.8 gActivationString3 Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationString3[] = "&did=";
```

Description

Third fragment of the activation URL. Continues until device ID is required.

5.4.9 gActivationString4 Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationString4[] = "&cvars=";
```

Description

Fourth fragment of the activation URL. Continues until the number of cumulative variables is required.

5.4.10 gActivationString5 Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationString5[] = "&rurl=http://";
```

Description

Fifth fragment of the activation URL. Continues until the return URL is required.

5.4.11 gActivationString6 Variable

File

CustomHTTPApp.c

C

```
ROM char gActivationString6[] = "/return.htm&snonce=";
```

Description

Sixth fragment of the activation URL. Continues until the security nonce is required. Contains the end of the return URL.

5.4.12 gCumulativePower Variable

File

MainDemo.c

C

```
INT64 gCumulativePower[3];
```

Description

Declare an INT64 to contain the cumulative power consumption of the device or circuit you're monitoring

5.4.13 goog_host Variable

File

gstatus.c

C

```
ROM BYTE goog_host[];
```

Description

Definition of the Google host name; used in data and status messages

5.4.14 gpkey_info Variable

File

ginsu.h

C

```
G_PKEY_INFO gpkey_info;
```

Description

Public Key Structure Information

5.4.15 gPowerMeterPreferences Variable

File

MainDemo.h

C

```
POWER_METER_PREFERENCES gPowerMeterPreferences;
```

Description

Declare POWER_METER_PREFERENCES (see page 32) structure to contain required settings

5.4.16 StackStartTime Variable

File

MainDemo.c

C

```
DWORD StackStartTime = 0;
```

Description

Declare a variable to hold the stack's relative start time

Index

—
 _g_pkey_info structure 30
 _pref_list enumeration 33

A

Additional Information 1
 AppConfig variable 37
 Application API 14
 APPLICATION_BUILD macro 34
 auth_path variable 37
 AUTH_PATH_STR enumeration member 33
 auth_token variable 38
 AUTH_TOKEN_STR enumeration member 33
 Authentication 7

C

CalculatePubKeyHash function 17
 CAPTURE_SEC_INT enumeration member 33
 captureMode variable 38
 Configuration and Programming 6
 ConvertUtcSecToString function 17
 Creating a Custom Application 11
 CustomHTTPApp 14

D

Development Tools 4
 DEVICE_MANUFACTURER macro 34
 DEVICE_MODEL macro 34
 DEVICE_NUM_SENSORS macro 35

F

Firmware Requirements 4
 Functions 14

G

G_BATCH_SEND_RESULT enumeration member 30
 G_BATCH_SEND_SUCCESS enumeration member 30
 G_CAPTURE_ATTEMPT enumeration member 30

G_CAPTURE_SUCCESS enumeration member 30
 G_ERROR_BAD_STATE enumeration member 30
 G_ERROR_BAD_VALUE enumeration member 30
 G_ERROR_NO_SEND_SPACE enumeration member 30
 G_ERROR_NO_SOCKET enumeration member 30
 G_ERROR_TIMEOUT enumeration member 30
 G_PKEY_INFO structure 30
 G_PUBKEY_ATTEMPT enumeration member 30
 G_PUBKEY_SUCCESS enumeration member 30
 G_SEND_ATTEMPT0 enumeration member 30
 G_SEND_ATTEMPT1 enumeration member 30
 G_SEND_ATTEMPT2 enumeration member 30
 G_SEND_RESULT enumeration member 30
 G_SEND_SUCCESS enumeration member 30
 G_STATUS_ATTEMPT enumeration member 30
 G_STATUS_RESULT enumeration member 30
 G_STATUS_SUCCESS enumeration member 30
 G_TOTAL_STATUS_COUNTERS enumeration member 30
 gActivationConfirm variable 38
 gActivationString1 variable 38
 gActivationString2 variable 39
 gActivationString3 variable 39
 gActivationString4 variable 39
 gActivationString5 variable 39
 gActivationString6 variable 40
 gCumulativePower variable 40
 GetBuildNumber function 18
 GetCumulativePower function 18
 GetHashListEntry function 19
 GetIpv4 function 19
 GetMac function 20
 GetPreferenceInt32 function 20
 GetPreferenceStr function 21
 GetSQ function 21
 GetSQDetails function 22
 Getting Help 2
 GetUptimeSec function 22
 GetUtcTimeSec function 23
 GetWebPort function 23
 GInitForWork function 24
 ginsu 16
 goog_host variable 40

GOOGLE_DATA_PORT macro 35
GOOGLE_HOST macro 35
GOOGLE_STATUS_PORT macro 35
gpkey_info variable 40
GPollForWork function 24
gPowerMeterPreferences variable 41
GQ_NUM_QUEUES macro 36
GSendInit function 24
GSTATUS_COUNTERS enumeration 30
gstatus_counters_type enumeration 30
GUpdateNVRam function 25

H

How the Application Works 10
How the Web Page Works 10
HTTPDeactivateDevice function 14
HTTPPostAuthenticate function 15
HTTPPostAuthInfo function 15

I

InitNVMemContents function 27
Introduction 1

M

Macros 33
MainDemo 27
MAX_NUM_PUBKEY_HASH macro 36

N

NV_MEM_STRUCTURE_OFFSETS enumeration 31

O

Operation 8

P

PadBuffer function 25
PMBUTTON_STATE enumeration 32
PMCaptureMode enumeration 32
POWER_METER_PREFERENCES structure 32
PREF_LIST enumeration 33
ProcessIO function 28

PUBKEY_HASH_SIZE macro 36

R

Running the Demos 4

S

SaveNVMemContents function 28
SavePowerMeterPreferences function 29
SEND_SEC_INT enumeration member 33
SEND_STATUS_INT enumeration member 33
SensorStillActive function 26
SetHashListEntry function 26
SetupCalculatePubKeyHash function 27
StackStartTime variable 41
Structures 29
SW License Agreement 3

U

Using the Code 10

V

Variables 36