Getting Started with The ThingWorx C SDK on the Raspberry PI

Setting up your first Raspberry PI can be a challenge. Combining it with setting up and using the ThingWorx C-SDK to expose some simple properties that can be found on this device is what will be covered in this discussion. This document was compiled as the author worked through this process. This procedure may change over time as the SDK evolves.

Audience

A ThingWorx developer who has had some exposure to the ThingWorx Composer and modeling. Having created an edge application using another SDK may also be useful since this document does not go into detail on the process of binding a remote things. You should have a Raspberry PI, a display and a keyboard already attached and a blank SD card (or Micro SD card). Having some experience with the unix command line shell and the C programming language would be helpful but examples will provide the specific commands required to complete this example.

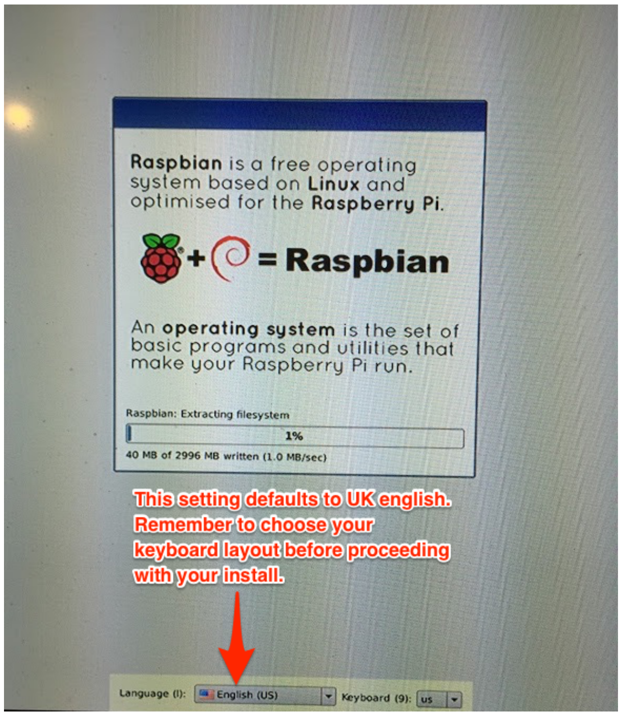
Contents

NOOBS is not just for noobs

Lets assume you have just purchased your first Raspberry PI and a blank SD card. You also have a USB mouse and keyboard attached and an adequate power supply (between 1.8 to 2.0 Amps at 5 volts). You have also hooked it up to an HDMI monitor (like a TV set). For help with these basic issues check out the "Getting Started" video at https://www.raspberrypi.org/help/quick-start-guide/.

NOOBS is an easy to use operating system installer for your PI. It can be downloaded at https://www.raspberrypi.org/downloads/noobs/ or by following the links on the "Getting Started" page. Download the NOOBS Offline and network install using your web browser. Once this download is complete, extract this zip file on your hard disk. Everything inside the expanded folder should be copied to your new, empty SD card. This process is described here: https://www.raspberrypi.org/help/noobs-setup/ . Once your have copied everything in this folder to your SD card you should eject it from your computer and insider it into the slot on the underside of your PI. Now plug in your power supply and NOOBS will configure your PI the first time it boots up.

An important thing to remember when viewing the noobs setup video is that the presenter is in the UK. If you are in the US or another country you should choose your native keyboard format before choosing to install the Raspberian Operating system.


Raspbian is a free operating system based on Linux and optimised for the Raspberry Pi.

+ = Raspbian

An operating system is the set of basic programs and utilities that make your Raspberry Pi run.

Raspbian: Extracting filesystem

1%

40 MB of 2996 MB written (1.0 MB/sec)

This setting defaults to UK english. Remember to choose your keyboard layout before proceeding with your install.

Language (l): English (US) Keyboard (9): us

Accessing Your Raspberry PI

Assuming everything has gone smoothly with your NOOBS install, you may already find yourself booted into the graphical user interface for your PI. If you have not, you may have to log from a command prompt and start the user interface manually. If asked for a username and password use the defaults which are:

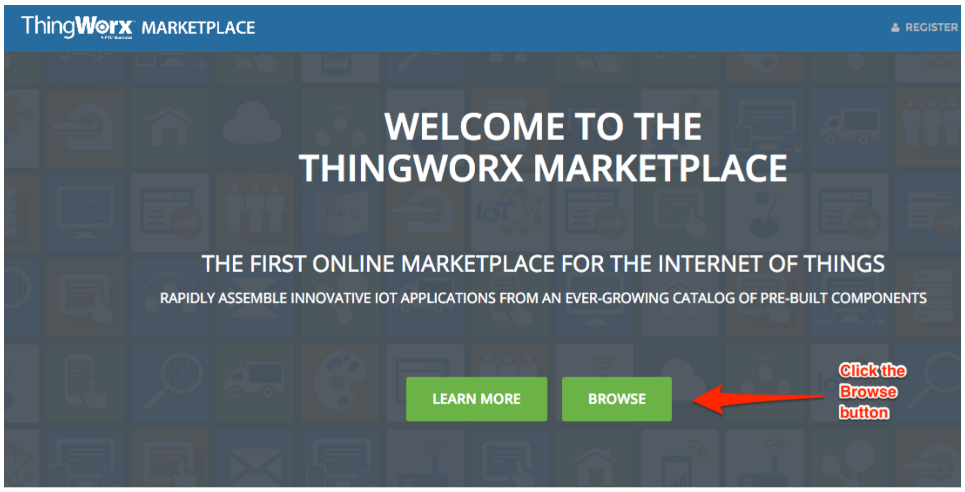
username: pi

password: raspberry

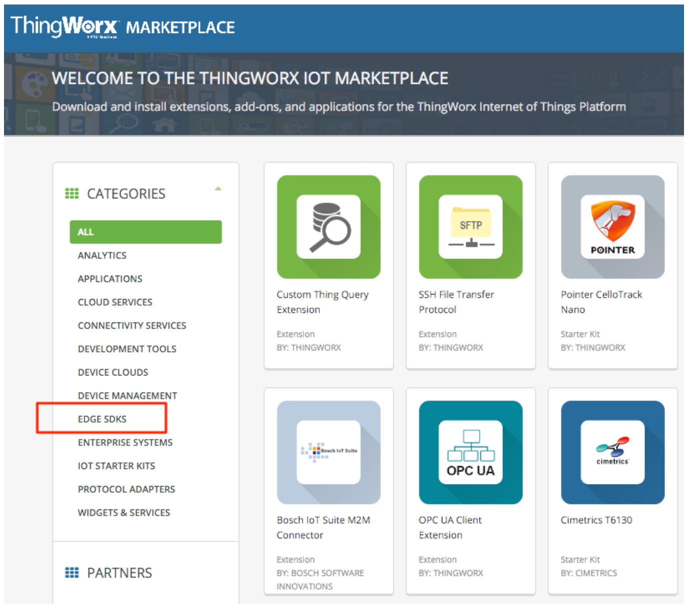
Once your are logged in, if the graphical user interface has not started type the command "startx" and hit return. Either way you should now be looking at an screen that looks similar to what you see below.

Getting the C SDK

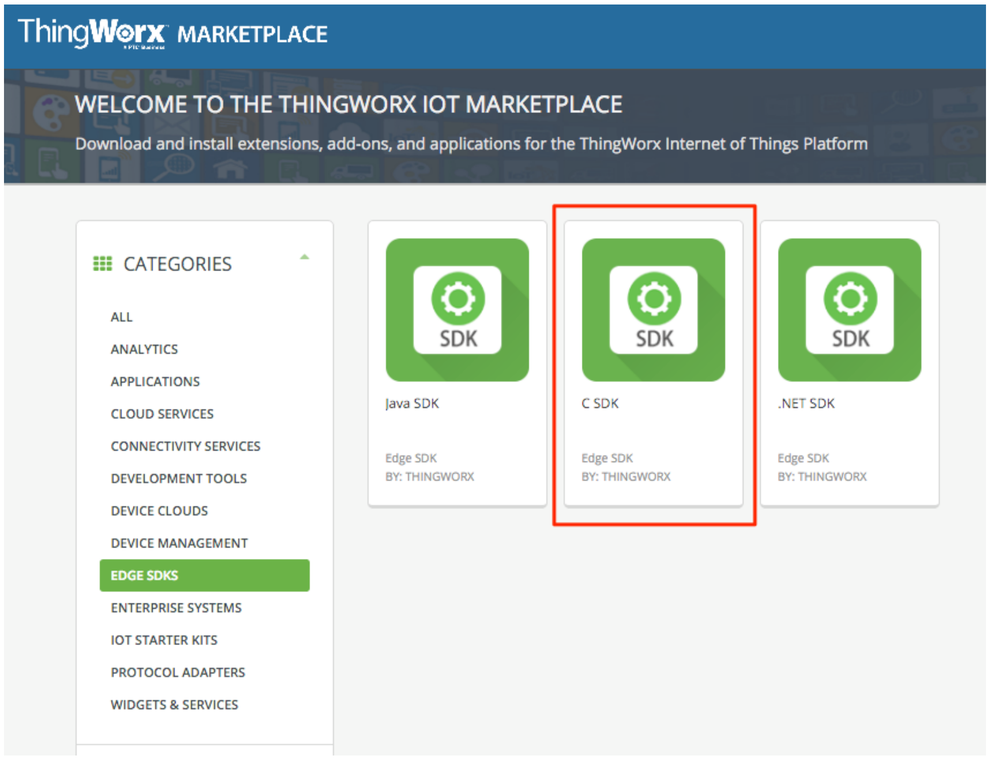
On a computer other than your PI, enter the URL, http://marketplace.thingworx.com/ to get to the ThingWorx Marketplace and select the "Browse" button.


ThingWorx MARKETPLACE
REGISTER
WELCOME TO THE THINGWORX MARKETPLACE
THE FIRST ONLINE MARKETPLACE FOR THE INTERNET OF THINGS
RAPIDLY ASSEMBLE INNOVATIVE IOT APPLICATIONS FROM AN EVER-GROWING CATALOG OF PRE-BUILT COMPONENTS
LEARN MORE
BROWSE
Click the Browse button

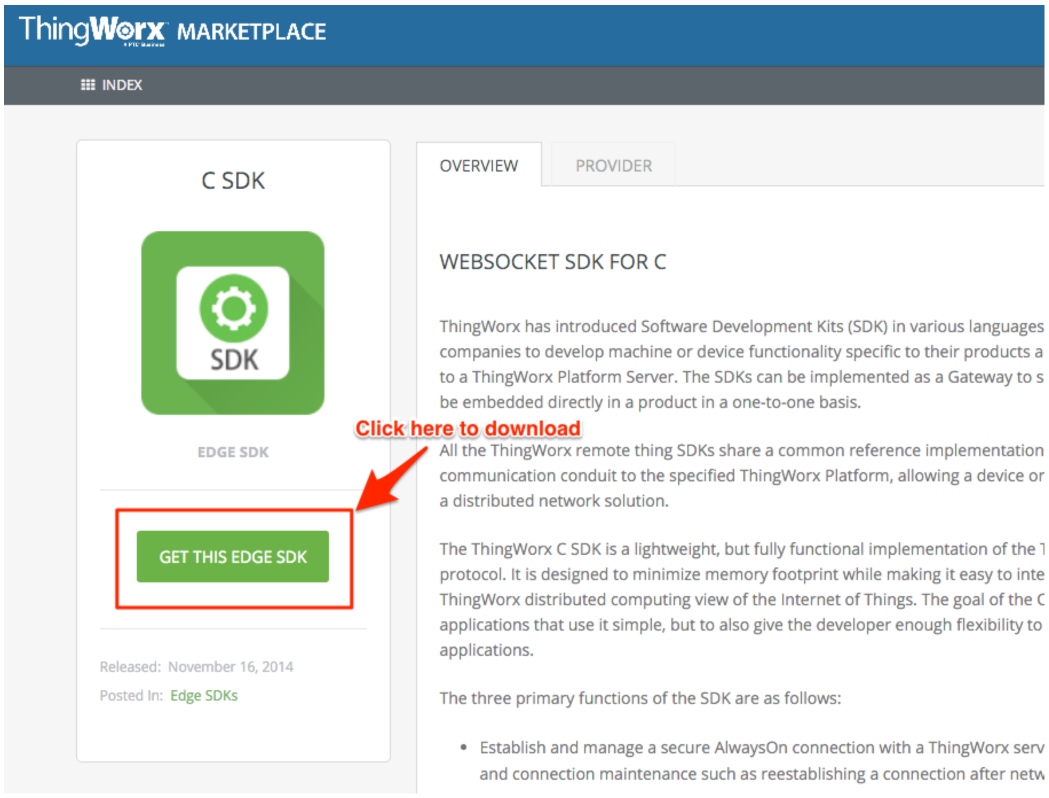
Now, select "Edge SDKs" as shown below.



Now select the C SDK.



Next, choose the download button. You will be asked to log into the marketplace. If you don't already have a log in, follow the instructions provided to get one.



WEBSOCKET SDK FOR C

ThingWorx has introduced Software Development Kits (SDK) in various languages
companies to develop machine or device functionality specific to their products a
to a ThingWorx Platform Server. The SDKs can be implemented as a Gateway to s
be embedded directly in a product in a one-to-one basis.

All the ThingWorx remote thing SDKs share a common reference implementation
communication conduit to the specified ThingWorx Platform, allowing a device or
a distributed network solution.

The ThingWorx C SDK is a lightweight, but fully functional implementation of the T
protocol. It is designed to minimize memory footprint while making it easy to inte
ThingWorx distributed computing view of the Internet of Things. The goal of the C
applications that use it simple, but to also give the developer enough flexibility to
applications.

The three primary functions of the SDK are as follows:

- Establish and manage a secure AlwaysOn connection with a ThingWorx serv
  and connection maintenance such as reestablishing a connection after netw

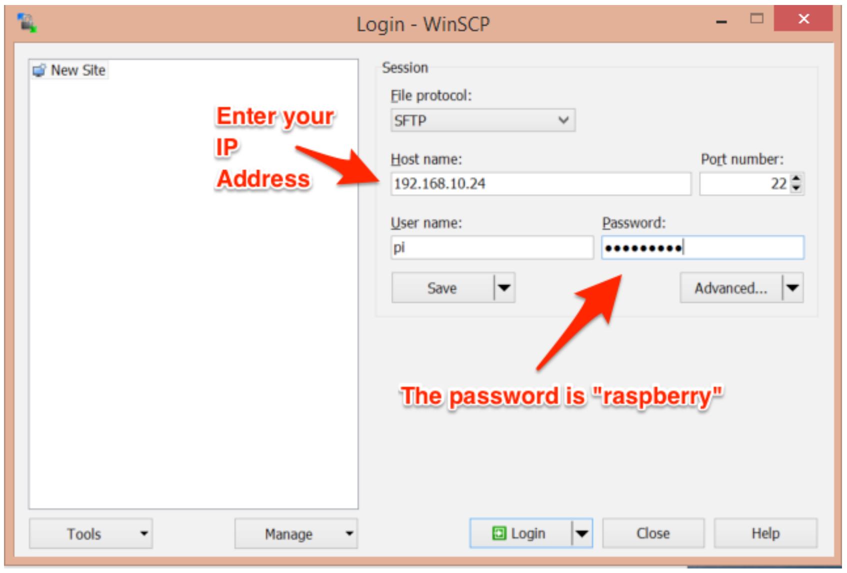
Once you have downloaded the C SDK, you will have to transfer it to your PI. The fastest way to do this in Windows is to use a program called winscp (https://winscp.net/eng/download.php). Choose the "Installation Package" and install. Once you have it running, you will need to find the IP address of your PI to proceed.

On your Raspberry PI type the ifconfig command to get your IP address:

```
pi@raspberrypi:~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:03:a2:31
          inet addr:192.168.10.24  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::4ebe:e9d6:4b89:4263/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:32162 errors:0 dropped:27 overruns:0 frame:0
          TX packets:12213 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14547528 (13.8 MiB)  TX bytes:7050031 (6.7 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:242 errors:0 dropped:0 overruns:0 frame:0
          TX packets:242 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18936 (18.4 KiB)  TX bytes:18936 (18.4 KiB)
```

Here the IP address is marked in red above. eth0 is the wired network connection. If you are using wireless you may have a wlan interface. Record this IP address for use in winscp. The winscp dialog is in the figure below. Once you fill in the fields hit the "Login" button. Say "Yes" to any "Add hosts to key cache messages".



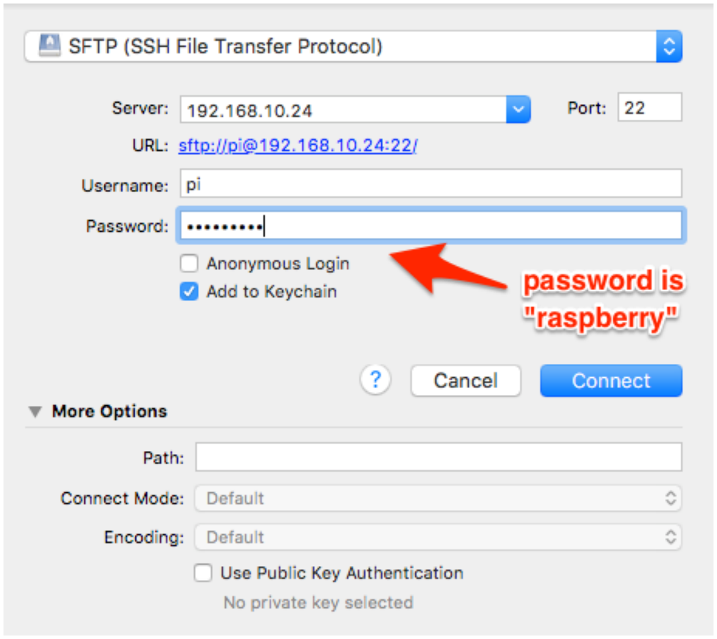
Enter your IP Address

File protocol: SFTP
Host name: 192.168.10.24
Port number: 22
User name: pi
Password: •••••••••

The password is "raspberry"

Once you are logged in, find your downloaded C SDK file which may look something like “MED-61061-CD-055_M010_ThingWorx-C-SDK-1-2-1-261-1.zip” on the left hand file list and drag it onto the right list which is the home directory of your PI.

If you use a Mac, the equivalent tool to winscp is called Cyberduck (https://cyberduck.io) . It will allow you to use SCP to transfer the C SDK to your PI in a very similar manner. Once installed, select File…>New Browser. Select the “Open Connection” button in the tool bar and fill in this dialog. Then hit the “Connect” button. You will then be able to drag the C-SDK zip file in a similar manner to winscp.


SFTP (SSH File Transfer Protocol)

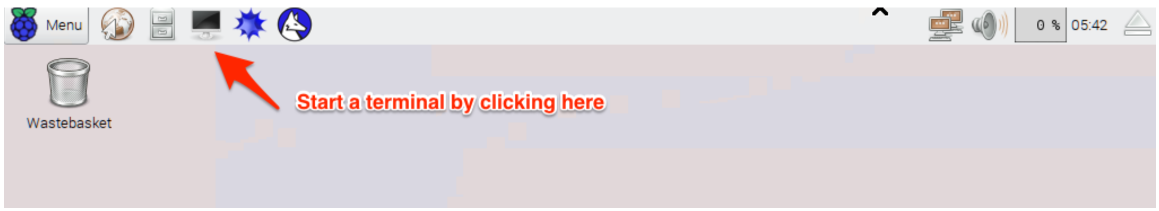
Server: 192.168.10.24 Port: 22
URL: sftp://pi@192.168.10.24:22/
Username: pi
Password: •••••••••
☐ Anonymous Login
☑ Add to Keychain

password is "raspberry"

More Options
Path:
Connect Mode: Default
Encoding: Default
☐ Use Public Key Authentication
No private key selected

Extracting the C SDK

Once you get the C SDK zip file into your PI things get easier. You will be working with the command line for these next steps so open one from the bar across the top of your PI’s user interface.



We will be working with the PI command line for most of the rest of this tutorial. If you are not experienced with the PI command line interface, try going over this unix tutorial here to get up to speed. http://www.ee.surrey.ac.uk/Teaching/Unix/ .

Lets make sure that your SDK file is on your PI. Type the command “ls” and hit enter.

```
pi@raspberrypi:~ $ ls

Desktop    Downloads            MED-61061-CD-055_M010_ThingWorx-C-
SDK-1-2-1-261-1.zip  Pictures   python_games  Videos

Documents  Music
```

Here you can see I have an SDK zip document called MED-61061-CD-055_M010_ThingWorx-C-SDK-1-2-1-261-1.zip. Lets unzip it with the command, unzip.

```
pi@raspberrypi:~ $ unzip MED-61061-CD-055_M010_ThingWorx-C-
SDK-1-2-1-261-1.zip

Archive:  MED-61061-CD-055_M010_ThingWorx-C-SDK-1-2-1-261-1.zip

   creating: tw-c-sdk/

   creating: tw-c-sdk/doc/

  inflating: tw-c-sdk/doc/mainpage.md……
```

The output goes on to list all the files being extracted. If we use the ls command again we now see…

```
pi@raspberrypi:~ $ ls

Desktop  Documents  Downloads  MED-61061-CD-055_M010_ThingWorx-C-
SDK-1-2-1-261-1.zip  Music  Pictures  Public  python_games  Templates
tw-c-sdk  Videos
```

Where you will see that a new directory called "tw-c-sdk" is now present. We now have a C SDK on our PI to work with.

The Steam Example

To get a feel for the process of exposing data to ThingWorx, lets compile an example that comes with the SDK. The Steam example exposes a few simple, simulated, properties to the server. We will start my looking in the tw-c-sdk/examples/ SteamSensor directory. Lets take a look at it in the terminal window.

```
pi@raspberrypi:~ $ cd tw-c-sdk

pi@raspberrypi:~/tw-c-sdk $ ls

build  doc  examples  src  version.txt

pi@raspberrypi:~/tw-c-sdk $ cd examples

pi@raspberrypi:~/tw-c-sdk/examples $ ls

SteamSensor                  SteamSensorWithFileTransferAndTunneling
SteamSensorWithOpenSSL                SteamSensorWithThreads

SteamSensorWithFileTransfer  SteamSensorWithMinimalFootprint
SteamSensorWithSubscribedProperties

pi@raspberrypi:~/tw-c-sdk/examples $ cd SteamSensor

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor $ ls

linux  osx  src  win32

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor $ cd src

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/src $ ls
```

```
main.c
```

The main.c file contains the steam sensor program which will create and update properties on the ThingWorx server.

You can view the program by typing the command “nano main.c”. Nano is a terminal text editor. When done you can exit nano with the ctrl-x command. You can also view this file with a graphical editor by using the command “idle main.c”.

Compiling the Steam Example

Next, we need to compile this program and at the same time, compile the C SDK targeting the linux arm platform for the Raspberry PI. Below is the terminal output for moving from the src directory to the linux directory where we will be running the linux “make” command to compile main.c into a binary executable program for the PI.

```
pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/src $ cd ..

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor $ cd linux

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux $ ls

Make.CommonSettings  Makefile

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux $ make
PLATFORM=gcc-linux-arm BUILD=release
```

Now that this is complete, the finished program can be found in the ~/tw-c-sdk/examples/SteamSensor/linux/bin/gcc-linux-arm/release directory.

```
pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux $ ls

bin  Make.CommonSettings  Makefile  obj

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux $ cd bin

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux/bin $ ls

gcc-linux-arm

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux/bin $ cd gcc-
linux-arm/

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux/bin/gcc-linux-arm
$ ls

release

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux/bin/gcc-linux-arm
$ cd release

pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux/bin/gcc-linux-arm/
release $ ls

SteamSensor
```

The compiled program is called “SteamSensor” and can be run with the command “./SteamSensor”. When you do this, you will see the following output which indicates that it cannot find a ThingWorx server.

```
pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/linux/bin/gcc-linux-arm/release
$ ./SteamSensor
[FORCE] 2015-12-26 06:19:06,257: Starting up
[DEBUG] 2015-12-26 06:19:06,257: twWs_Create: Initializing Websocket Client for
localhost:443//Thingworx/WS
[DEBUG] 2015-12-26 06:19:06,257: twTlsClient_Create: Initializing TLS Client
[DEBUG] 2015-12-26 06:19:06,261: subscribedPropsMgr_Initialize: Initializing
subscribed properties manager
[TRACE] 2015-12-26 06:19:06,261: twApi_Connect: Delaying 1 milliseconds before
connecting
[DEBUG] 2015-12-26 06:19:06,263: twTlsClient_Reconnect: Re-establishing SSL
context
[DEBUG] 2015-12-26 06:19:06,264: twTlsClient_Connect: Connecting to server
[ERROR] 2015-12-26 06:19:06,264: Error intializing socket connection.  Err =
111
```

The program can be ended by hitting ctrl-c or typing the letter q.

Configuring the Steam Example

Changing the Hostname, AppKey and Port

This tutorial assumes you are running an instance of ThingWorx on port 8080 of some host over HTTP. Be default this example assumes you are running on localhost using HTTPS. We will have to change this setting. To make these changes we must edit tw-c-sdk/examples/SteamSensor/src/main.c and change some values.

```
pi@raspberrypi:~/tw-c-sdk/examples/SteamSensor/src $ nano main.c
```

Find the section near the top of the file that looks like this:

```
/* Server Details */
#define TW_HOST "localhost"
#define TW_APP_KEY "e9274d87-58aa-4d60-b27f-e67962f3e5c4"
```

And replace localhost with the hostname of your server. Remember, you can't use localhost because your ThingWorx server is not located on your PI. You must know either its hostname or IP address and enter it here.

You will also have to change the App Key to a key that exists on your server. Look in the Application Keys section of your server and for this example, choose or create an app key that has Administrator privileges.

Next we may have to change the port your ThingWorx server is on. Look for a line in this file that looks like this:

```
#if defined NO_TLS
        int16_t port = 80;
#else
```

```
        int16_t port = 443;
#endif
```

You may have to change the port variable. For example, if your ThingWorx server is on port 8080, then change the NO_TLS port to 8080.

Now save this file with ctrl-x and then hit "Y".

Disabling HTTPS (If Required)

If you intend to use HTTP instead of HTTPS on your ThingWorx server you must edit the twLinux.h file located in tw-c-sdk/src/porting/twLinux.h. You must add the text "#define NO_TLS" to this file and change the TW_TLS_INCLUDE define to "twNoTls.h" as shown below in red:

```
/**
 * \brief Define which pluggable TLS library is used.
 *
 * \note The NO_TLS option turns off encryption altogether which may be
useful
 * for debugging but is also <b>not recommended for production
environments</b>
 * as it may introduce serious security risks.
*/
#define NO_TLS
#define TW_TLS_INCLUDE "twNoTls.h"
```

Note:

If you are using HTTPS on your ThingWorx server but you are using a self signed certificate you would not make the changes above to disable HTTPS support. You will have to edit the file tw-c-sdk/src/tls/twTls.c to disable certificate validation. In this file modify the existing line shown in red. This value must be changed from TRUE to FALSE.

```
tls->options = options;
tls->selfSignedOk = 0;
tls->validateCert = FALSE;
tls->isEnabled = TRUE;
/*  Create our socket */
tls->connection = twSocket_Create(host, port, 0);
```

You have now disabled HTTPS support and can connect to an HTTP server if required.

now cd to "tw-c-sdk/examples/SteamSensor/linux" and re-run the "make PLATFORM=gcc-linux-arm BUILD=release" command again to rebuild your SteamSensor. Now "cd bin/gcc-linux-arm/release/" and run "./SteamSensor" again.

You should now be connected to your server. You can verify this by going to your server, pulling down the "Monitoring: menu and selecting "Remote Things". You should now see a "SteamSensor1" in the "Unbound" tab. It would only move to the "Bound" tab if a Thing with the same name using the "RemoteThing" template was created.

Creating your own, Custom Application

Now that you have gotten the SteamSensor application running, we can use it as a template to create our own custom application that will read the current board temperature of the Raspberry PI and relay it to the ThingWorx server.

Start by copying the current SteamSensor example to a new directory:

```
cd ~/tw-c-sdk/examples

cp -a SteamSensor PIDataCollector
```

This will duplicate the SteamSensor example in a new directory called PIDataCollector. Now we will change the name of the executable program it will produce when build.

```
cd PIDataCollector/linux

nano Makefile
```

Now change the value of TW_APP_NAME variable to PIDataCollector and save this file.

```
################## APP SPECIFIC DEFINES #######################

# TW_APP_NAME is the desired name of the application executable

TW_APP_NAME      = PIDataCollector
```

Writing Your Own Program

Replace your main.c in the ~/tw-c-sdk/examples/PIDataCollector/src directory with the source code below. This example will read your current board temperature and relay it to ThingWorx. You should have a copy of the new main.c and a Things_PIDataCollector1.xml file you can import into ThingWorx to bind this thing with on the server.

```
/**

 * This example program reads the current board temperature of a
Raspberry PI

 * and delivers it to a ThingWorx server.

 */
```

```
#include "twOSPort.h"
#include "twLogger.h"
#include "twApi.h"

#include <stdio.h>
#include <string.h>

/* Name of the thing you are binding to. */
char * thingName = "PIDataCollector1";

/* Server Details */
#define TW_HOST "your.thingworxserver.com" // Set this to your
ThingWorx server name
#define TW_APP_KEY "e9274d87-58aa-4d60-b27f-e67962f3e5c4" // Change
this to a appKey from your server

/* A simple structure to store current property values. */
struct  {
    double BoardTemperature;
    char * DeviceName;
} properties;

/**
 * Reads the current board temperature of this raspberry PI from the /
sys tree.
 */
double getBoardTemperature(){
    FILE *temperatureFile;
    double T;
    temperatureFile = fopen ("/sys/class/thermal/thermal_zone0/temp",
"r");
    if (temperatureFile == NULL) {
        TW_LOG(TW_FORCE,"device file /sys/class/thermal/thermal_zone0/
temp failed to open.");
        return 0;
    }
    fscanf (temperatureFile, "%lf", &T);
    T /= 1000;
    T = T * 1.8f + 32;
    //printf ("The temperature is %6.3f F.\n", T);
    fclose (temperatureFile);
    return T;
}

/**
 * Sends property values stored on your PI back to the ThingWorx
server. You must
 *  add your properties to this list for them to be updated on the
server.
 */
void sendPropertyUpdate() {

      /* Create the property list */
      propertyList * proplist =
twApi_CreatePropertyList("BoardTemperature",twPrimitive_CreateFromNumbe
r(properties.BoardTemperature), 0);
      if (!proplist) {
            TW_LOG(TW_ERROR,"sendPropertyUpdate: Error allocating
property list");
            return;
      }

twApi_AddPropertyToList(proplist,"DeviceName",twPrimitive_CreateFromStr
ing(properties.DeviceName,TRUE), 0);
      twApi_PushProperties(TW_THING, thingName, proplist, -1, FALSE);
      twApi_DeletePropertyList(proplist);
}

/**
 * Called every DATA_COLLECTION_RATE_MSEC milliseconds, this function
is responsible for polling
 * and updating property values.
 */
#define DATA_COLLECTION_RATE_MSEC 2000
void dataCollectionTask(DATETIME now, void * params) {
    properties.BoardTemperature = getBoardTemperature();
      sendPropertyUpdate();
}
```

```
/**
 * This function is responsible for processing read and write requests
from the server for the BoardTemperature property.
 */
enum msgCodeEnum propertyHandlerBoardTemperature(const char *
entityName, const char * propertyName,  twInfoTable ** value, char
isWrite, void * userdata) {
    if (value) {
        if (isWrite && *value) {
            // Value is being set from the server
            // We are not supporting writes to this property
        } else {
            // Value is being read from the server
            *value = twInfoTable_CreateFromNumber(propertyName,
properties.BoardTemperature);
        }
        return TWX_SUCCESS;
    } else {
        TW_LOG(TW_ERROR,"Error updating value");
    }
    return TWX_BAD_REQUEST;
}


/**
 * This function is responsible for processing read and write requests
from the server for the DeviceName property.
 */
enum msgCodeEnum propertyHandlerDeviceName(const char * entityName,
const char * propertyName,  twInfoTable ** value, char isWrite, void *
userdata) {
    if (value) {
        if (isWrite && *value) {
            // Value is being set from the server
            // We are not supporting writes to this property
        } else {
            // Value is being read from the server
             *value = twInfoTable_CreateFromString(propertyName,
properties.DeviceName, TRUE);
        }
        return TWX_SUCCESS;
    } else {
        TW_LOG(TW_ERROR,"Error updating value");
        return TWX_BAD_REQUEST;
    }
}

/**
 * This is the main entry point of this application.
 * It is responsible for establishing a ThingName, registering
properties
 * and establishing your connection to the servers.
 */
int main( int argc, char** argv ) {

#if defined NO_TLS
      int16_t port = 8080;
#else
      int16_t port = 443;
#endif
      int err = 0;
      DATETIME nextDataCollectionTime = 0;

      twLogger_SetLevel(TW_TRACE);
      twLogger_SetIsVerbose(1);
      TW_LOG(TW_FORCE, "Starting up");

      /* Initialize Properties */
    properties.BoardTemperature = 0.0;
    properties.DeviceName = "My Raspberry PI";

      /* Initialize the API */
      err = twApi_Initialize(TW_HOST, port, TW_URI, TW_APP_KEY, NULL,
MESSAGE_CHUNK_SIZE, MESSAGE_CHUNK_SIZE, TRUE);
      if (err) {
            TW_LOG(TW_ERROR, "Error initializing the API");
            exit(err);
      }

      /* Regsiter our properties */
    twApi_RegisterProperty(TW_THING, thingName, "DeviceName",
TW_STRING, NULL, "ALWAYS", 0, propertyHandlerDeviceName, NULL);
    twApi_RegisterProperty(TW_THING, thingName, "BoardTemperature",
TW_NUMBER, NULL, "ALWAYS", 0, propertyHandlerBoardTemperature, NULL);


      /* Bind our thing - Can bind before connecting or do it when the
onAuthenticated callback is made.  Either is acceptable */
      twApi_BindThing(thingName);


      /* Connect to server */
      err = twApi_Connect(CONNECT_TIMEOUT, twcfg.connect_retries);
      if (!err) {
            /* Register our "Data collection Task" with the tasker */
            twApi_CreateTask(DATA_COLLECTION_RATE_MSEC,
dataCollectionTask);
      } else {
            TW_LOG(TW_ERROR,"main: Server connection failed after %d
attempts.  Error Code: %d", twcfg.connect_retries, err);
      }


      while(TRUE) {
            DATETIME now = twGetSystemTime(TRUE);
            twApi_TaskerFunction(now, NULL);
            twMessageHandler_msgHandlerTask(now, NULL);
            if (twTimeGreaterThan(now, nextDataCollectionTime)) {
                  dataCollectionTask(now, NULL);
                  nextDataCollectionTime = twAddMilliseconds(now,
DATA_COLLECTION_RATE_MSEC);
            }
            twSleepMsec(5);
      }
      twApi_UnbindThing(thingName);
      twSleepMsec(100);


      twApi_Delete();
      exit(0);
}
```

Now you can build and run the PIDataCollector.

```
cd ~/tw-c-sdk/examples/PIDataCollector/linux
make PLATFORM=gcc-linux-arm BUILD=release
./bin/gcc-linux-arm/release/PIDataCollector
```

You can quit either by hitting the q key or ctrl-c.

Conclusion

This example is meant as a demonstration of how to use the ThingWorx C SDK to create a simple C command line application that delivers real data from the Raspberry PI's own internal on board thermometer to ThingWorx. It also illustrates how easy it is to create your own custom C program to deliver your own data to ThingWorx. This example only takes advantage of the most basic features of the C SDK. To learn more you should take a closer look at the SteamSensor example and also the documentation that comes with the C SDK.

Written by William Reichardt, Software Engineer, Thingworx (2015)