Samsung ARTIK Advanced Developer Guide for Thingworx integration

Version 2.0

Introduction

From the basic setup guide, you should have learned how to easily adapt a Thingworx C sample application on ARTIK platform and use it to stream data to ThingWorx server. In this guide, we will show you how you can create your own, custom application.

We will use the SteamSensor application as a template to create our own custom application that reads distance sensor data from ARTIK and relays it to the ThingWorx server.

Start by copying the SimpleThingClient example to a new directory:

```
cd ~/tw-c-sdk/examples
cp –a SimpleThingClient ARTIK
```

Now, we will change the name of executable program it generates when compiling.

```
cd ARTIK/linux
vi Makefile
```

Change the value of TW_APP_NAME to ARTIK and save the file.

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

# TW_APP_NAME is the desired name of the application executable
TW_APP_NAME      = ARTIK
```

Custom the application

Open main.c, replace TW_HOST and TW_APP_KEY with the hostname of your server and the application key you created.

Then, we will define our thing name and edit properties data structure to include more properties.

```
/* Name of our thing */
char * thingName = "ARTIKThing";

/*****************
  A simple structure to handle
  properties. Not related to
  the API in anyway, just for
  the demo application.
******************/
 struct  {
   char * artikID;
   twLocation location;
   int distance;
   int distanceLimit;
 } properties;
```

Collecting sensor data

We use an ARTIK 530 for this example and connect the temperature sensor to analog pin 0. In order to read data from the pin, we need to add the function below to main.c.

```
int pin=0;
/**
* Read the temperature sensor data from ARTIK
*/
int getTemperature() {
  FILE * fd;
  char fName[64];
  char val[8];

  // open value file
  sprintf(fName,
"/sys/devices/platform/c0000000.soc/c0053000.adc/iio:device0/in_vol
tage%d_raw", pin);
  if ((fd = fopen(fName, "r")) == NULL) {
    printf("Error: can't open analog voltage value\n");
    return 0;
  }
  fgets(val, 8, fd);
  fclose(fd);

  return atoi(val);
}
```

Updating properties in the ARTIK Example

Next, we need to adapt `main.c` code to support our new properties.
1. In `main()` function, look for the lines that register properties by using twApi_RegisterProperty(). Update this part to the code below:

```
   /* Regsiter our properties */
    twApi_RegisterProperty(TW_THING, thingName, "artikID",
TW_STRING, NULL, "ALWAYS", 0, propertyHandler, NULL);
    twApi_RegisterProperty(TW_THING, thingName, "location",
TW_LOCATION, NULL, "ALWAYS", 0, propertyHandlerDistance, NULL);
    twApi_RegisterProperty(TW_THING, thingName,
"distance",TW_INTEGER, NULL, "ALWAYS", 0, propertyHandler, NULL);
    twApi_RegisterProperty(TW_THING, thingName, "distanceLimit",
TW_INTEGER, NULL, "ALWAYS", 0, propertyHandler, NULL);
```

2. `sendPropertyUpdate()` is the function where you re-construct your list of properties before sending it to Thingworx Server.

```
void sendPropertyUpdate() {
```

```
  TW_LOG(TW_INFO, "send Property Update");
  propertyList * proplist = twApi_CreatePropertyList("distance",
twPrimitive_CreateFromInteger(properties.distance), 0);
  twApi_AddPropertyToList(proplist, "artikID",
twPrimitive_CreateFromString(properties.artikID,TRUE), 0);

twApi_AddPropertyToList(proplist,"location",twPrimitive_CreateFromL
ocation(&properties.location), 0);
  twApi_AddPropertyToList(proplist, "distanceLimit",
twPrimitive_CreateFromInteger(properties.distanceLimit), 0);

  twApi_PushProperties(TW_THING, thingName, proplist, -1, FALSE);
  twApi_DeletePropertyList(proplist);
}
```

3. `sendPropertyUpdate()` function is being invoked from `dataCollectionTask()` , who is responsible for polling and updating property values. We also include local logic to check if the sensor reading reaches the threshold value defined in `properties.distanceLimit`.

```
/**
* Called every DATA_COLLECTION_RATE_MSEC millseconds, this function
is
* responsible for polling and updating property values.
*/
#define DATA_COLLECTION_RATE_MSEC 2000
void dataCollectionTask(DATETIME now, void * params) {
  properties.distance = getDistance();
  printf("get Distance=%d\n", properties.distance);

  /* Check for an alert */
  if (properties.distance > properties.distanceLimit) {
    twInfoTable * faultData = 0;
    char msg[100];
    sprintf(msg, "%s distance value %d exceeds threshold of %d",
thingName, properties.distance, properties.distanceLimit);
    printf(msg);
    faultData = twInfoTable_CreateFromString("message", msg, TRUE);
    twApi_FireEvent(TW_THING, thingName, "DistanceSensorFault",
faultData, -1, TRUE);
    twInfoTable_Delete(faultData);
  }

  /* Update the properties on the server */
  sendPropertyUpdate();
}
```

4. `propertyHandler()` is the callback function that reads and writes requests from the server.

```
/**
```

```
* This function processes read and write requests from the server
*/
enum msgCodeEnum propertyHandler(const char * entityName, const
char * propertyName, twInfoTable ** value, char isWrite, void *
userdata) {
  TW_LOG(TW_INFO, "propertyHandler - Function called for Entity %s,
property %s", entityName, propertyName);

  if (value) {
    if (isWrite && *value) {
      //
      if(strcmp(propertyName, "distanceLimit") == 0) {
         twInfoTable_GetInteger(*value, propertyName, 0,
&properties.distanceLimit);
         twApi_SetSubscribedProperty(entityName, propertyName,
twPrimitive_CreateFromNumber(properties.distanceLimit), FALSE,
TRUE);
      }
    } else {
      //Property Reads
      if (strcmp(propertyName, "distance") == 0)
         *value = twInfoTable_CreateFromInteger(propertyName,
properties.distance);
      else if (strcmp(propertyName, "artikID") == 0)
         *value = twInfoTable_CreateFromString(propertyName,
properties.artikID,TRUE);
      else
         return TWX_NOT_FOUND;
    }
    return TWX_SUCCESS;
  } else {
    TW_LOG(TW_ERROR, "Error updating value");
  }
  return TWX_BAD_REQUEST;
}
```

Initiating properties in the ARTIK Example

In `main()` function, look for the part to initialize properties and initialize as below:

```
/* Initialize Properties */
properties.artikID = "ARTIK 530";
properties.location.longitude = 37.4094279;
properties.location.latitude = -121.94621940000002;
properties.distanceLimit = 2000;
```

Compiling the ARTIK Example

Next, we need to compile the program and give it a try. Below is the terminal output for moving from the *src* directory to *linux* directory where we will be running the linux "*make*"command to compile the C files into a binary executable for the ARTIK.

```
[root@artik src]# ls
main.c  simple_thing.c  simple_thing.h
[root@artik src]# cd ..
[root@artik ARTIK]# cd linux
[root@artik linux]# ls
bin  Make.CommonSettings  Makefile  obj  run_example.sh
[root@artik 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/ARTIK/linux/bin/gcc-linux-arm/release* directory.

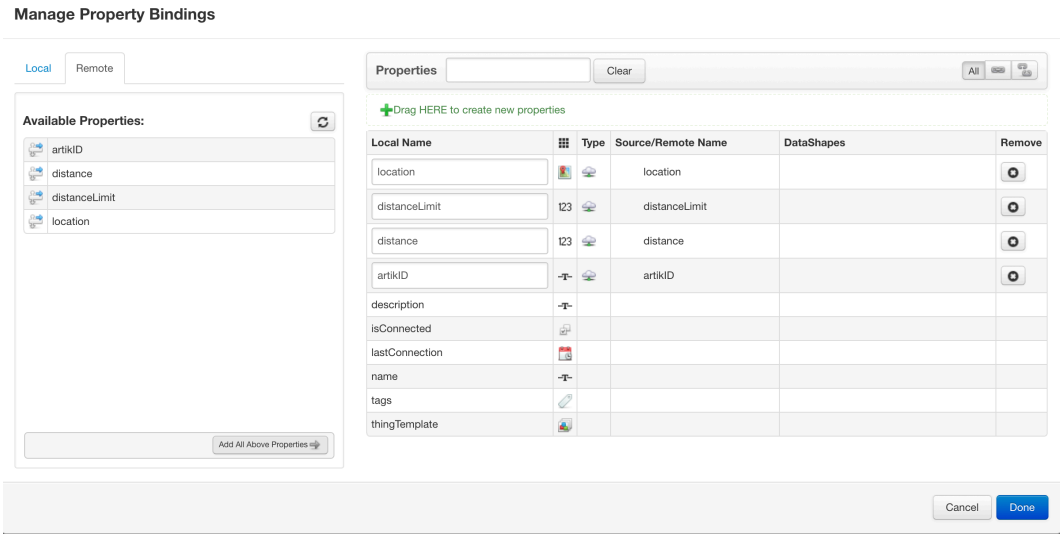
Creating *ARTIKThing* on your ThingWorx server by using *RemoteThing* template, then launch our application.

```
[root@artik linux]# ./bin/gcc-linux-arm/release/ARTIK
[FORCE] 2017-11-15 15:52:50,786: Starting up
[INFO ] 2017-11-15 15:52:50,952: twWs_Connect: Websocket connected!
[WARN ] 2017-11-15 15:52:51,152: twBindBody_Delete: NULL body or
stream pointer
[FORCE] 2017-11-15 15:52:51,252: AuthEventHandler: Authenticated using
appKey = 97da0578-0ff9-4a01-90c8-ac34bc3f7308.  Userdata = 0x0
[FORCE] 2017-11-15 15:52:51,351: BindEventHandler: Entity ARTIKThing
was Bound
```

Viewing the streamed data from ThingWorx Portal

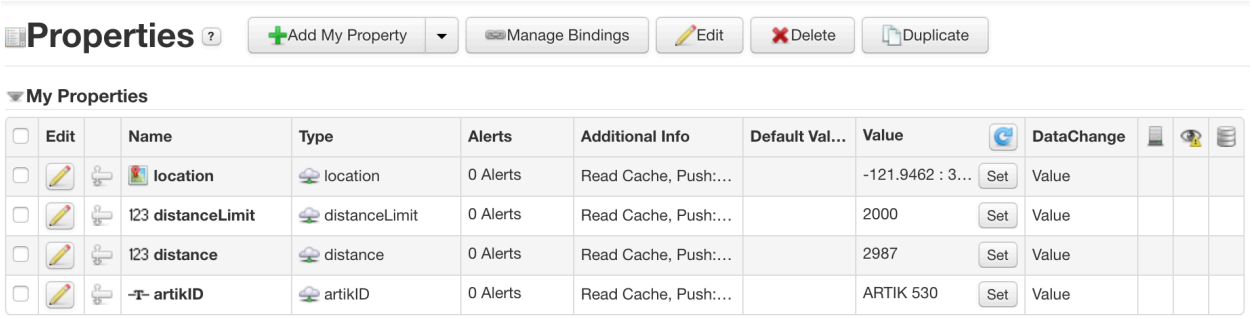
Select the **Properties** tab of *ARTIKThing* Thing. Here you should see the *isConnected* property is set to **true**, indicating that the connection is successful.

Click the **Manage Bindings** button on **Properties** tab to bind the properties. Select the Available Properties from Remote Tab, and drag them to the right side to create new properties.



Click Done to close Manage Property Bindings window, and Save your changes.

Now, from *ARTIKThing* Properties window, you should be able to see the properties being propagated from your ARTIK device to Thingworx.



Edit		Name	Type	Alerts	Additional Info	Default Val...	Value	DataChange
		location	location	0 Alerts	Read Cache, Push:...		-121.9462 : 3... Set	Value
		123 distanceLimit	distanceLimit	0 Alerts	Read Cache, Push:...		2000 Set	Value
		123 distance	distance	0 Alerts	Read Cache, Push:...		2987 Set	Value
		artikID	artikID	0 Alerts	Read Cache, Push:...		ARTIK 530 Set	Value

For subscribed property, if you click the Set button next to your property, and change its value, the updated value will be sent back to your ARTIK device.