TMG399x Reference Code Programmer’s Guide

Table of Contents

Revision History

Revision	Date	Owner	Description
1.0	2014.06.05		Initial Release

1 General Description

This document describes the programming interface for version 2.0.8 of the reference software for use with the ams TMG399x family of ALS-Color-Proximity-Gesture-IRBeam devices.

An understanding of the hardware operation of the TMG399x is necessary but is outside the scope of this document. Refer to the datasheet for detailed hardware descriptions.

1.1 Reference Code Content

The reference code consists of three basic portions: a Linux kernel driver, a Linux user space library to decode gestures, and a demo-only server program which demonstrates how to transfer data from the kernel driver into the library and then reports the results, either as a textual log, or by creating Linux keyboard events to simulate the detected gestures. It has been tested on three different Android platforms. It is expected that it can equally well be used on a pure Linux platform, though ams has not tested it that way.

2 Evaluation Platforms

The code has been built for, and successfully tested on, the following platforms:

- Arndale Board (Samsung Exynos 5250 SoC) running Android 4.1.1
- Beaglebone Black (TI Sitara AM335x SoC) running Android 4.2.2
- DragonBoard (Qualcomm Snapdragon 8074 SoC) running Android 4.2.2

The Beaglebone Black is recommended as the primary standard development board. This reference code release includes all the files needed to operate on both the Beaglebone Black and the Arndale. The DragonBoard implementation requires some additional files from Intrinsyc, the board maker, and therefore is not entirely supported using only the release.

3 Software Description

The basic arrangement of the software is shown in Figure 1. A Linux kernel driver, registered as an input driver, comes up via the standard .probe() etc. sequence under the control of the Linux input subsystem. The user space server is started either manually or via a Linux automatic launching facility of your choice, then communicates with the driver via Application Binary Interfaces (ABIs) in the form of sysfs files (special files in the /sys tree). The ABIs provided by the driver are described in a separate document - TMG399x Device Driver v2.0.8 ABI Descriptions

The main focus of the demo server is gestures. Detected gestures are reported as Linux keyboard input events and are thus available to all Linux and Android apps. In the demo server, Ambient Light Sensing (ALS) output is written to a log file and optionally printed to the console but is not propagated up to Android apps. Proximity output is only written to the log file. The driver supports IRBeam functionality but the demo server does not.

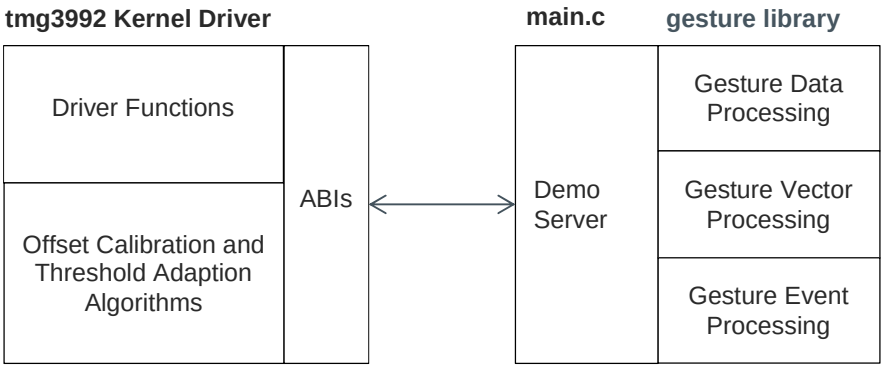

Figure 1 Software Architecture

4 Data Types Used

To interface with the kernel and the gesture library, a program requires several basic data types, derived from standard Linux definitions and several enumerations and structures which are defined by the gesture library code.

4.1 Basic Data Types

The basic data types used to interface with the kernel driver and gesture library are derived from standard Linux definitions. The types listed in the following table are part of the standard Linux distribution:

Type Name	Description	Source
int32_t	A 32-bit integer value	<stdint.h>
struct timespec	A structure for storing accurate time values.	<time.h>

For convenience, the glcommon.h file defines additional basic types that are used by all of the gesture library code:

Type Name	Declaration
i32	typedef int32_t i32;
u8	typedef unsigned char u8;
DateTime_t	typedef struct timespec DateTime_t;
TimeSpan_t	typedef struct timespec TimeSpan_t;

4.2 Gesture Data Structures and Definitions

The following data structures are defined and used by the gesture library code.

4.2.1 NSWE_t

The NSWE_t structure is used to store a single set of North/South/West/East values from the TMG399x hardware. Although the hardware only produces 8-bit values, for ease of use, they are stored and passed as 32-bit integers. This structure is declared in glcommon.h.

```
typedef struct
{
    i32 North;  /* north gesture data */
    i32 South;  /* south gesture data */
    i32 West;   /* west gesture data */
    i32 East;   /* east gesture data */
} NSWE_t;
```

4.2.2 gestureRawDataState_t

gestureRawDataState_t is an enumeration that the threshold adaption code in the kernel driver uses to communicate the state of the data collection to the gesture library. For various historical reasons, some of these values are obsolete and no longer used, but are maintained so that the actual enumeration values remain unchanged. This enumeration is declared in glcommon.h.

```
typedef enum
{
    idle,     /* no gesture processing active */
    started,  /* gesture collection continuing (already started) */
    tEntry,   /* New gesture collection has begun */
    entry,    /* unused */
    tEnded,   /* collection of current gesture has ended */
    ended     /* unused */
} gestureRawDataState_t;
```

4.2.3 adaptiveRawData_t

adaptiveRawData_t is part of the data structure that is passed out of the kernel mode driver. It is a subset of the data that is used by the gesture library. This structure is declared in glcommon.h.

```
typedef struct
{
    gestureRawDataState_t state;  /* state indication */
    NSWE_t NSWE;                  /* raw gesture data */
    i32 GProx;                    /* gesture prox value */
    i32 Count;                    /* counter value of gesture packet */
} adaptiveRawData_t;
```

4.3 gestureRawData_t

gestureRawData_t is a superset of the adaptiveRawData_t data structure. When passing the data to the gesture library, the user program must copy the existing data from the adaptiveRawData_t structure into the gestureRawData_t structure and fill in the additional time value. This structure is declared in glcommon.h.

```
typedef struct
{
    gestureRawDataState_t State;  /* state indication */
    DateTime_t Time;              /* timestamp for this sample */
    NSWE_t NSWE;                  /* raw gesture data */
    i32 GProxMax;                 /* gesture prox value */
    i32 Count;                    /* counter value of gesture packet */
} gestureRawData_t;
```

4.3.1 RawDataArray_t

RawDataArray_t is passed from the gesture library when a gesture has been detected. This structure contains all of the data that was collected for the gesture and certain summary information that applies to the raw data. This structure is declared in glcommon.h.

```
#define RAW_DATA_ARRAY_LENGTH  100

typedef struct
{
    DateTime_t Start_Time;  /* time of 1st point in gesture */
    i32 Peak_GProx_Index;   /* gprox value at gesture peak */
    i32 Count;              /* Num of datapoints rcvd (can be >100) */
    i32 Length;             /* Real length of RawData array (<=100) */
    NSWE_t NSWE_Offset;     /* DC Offsets applied to this gesture */
    NSWE_t NSWE_Scale100;   /* Scale value * 100 for this gesture */
    gestureRawData_t RawData[RAW_DATA_ARRAY_LENGTH];  /* all points */
    i32 Next_Long_Time;     /* next gesture button push time */
    i32 Long_Count;         /* number of long counts */
} RawDataArray_t;
```

The data in the RawDataArray_t structure can be used to determine the type and characteristics of the detected gesture. This can then be used to generate system events to report the gestures or to simply display the gesture information in a log message.

4.3.2 algStatusData_t

algStatusData_t is the complete data structure which is passed from the kernel driver to the user program via the gesture_data ABI. This structure is declared in main.c

```
typedef struct
{
    adaptiveRawData_t gRawData;
    bool gRawDataValid;          /* true = pass data to library */
    u8 prox_entry_baseline10;   /* baseline value used in calc'ing */
                                /* entry threshold, in tenths */
    u8 prox_entry_stdev10;      /* std. dev. value used in calc'ing */
                                /* entry threshold, in tenths */
    u8 prox_entry_threshold;    /* gesture prox entry threshold */
    u8 gesture_exit_threshold;  /* gesture exit threshold */
    NSWE_t gesture_offset;      /* DC offsets applied to raw data */
    bool doOffsetCalibration;   /* true = offset cal should be done */
    bool doGestureDcInit;       /* internal flag for gesture cal */
    bool doProxDcInit;          /* internal flag for DC cal */
} algStatusData_t;
```

5 Interfacing with the Kernel Driver

This document will not deal with the specifics of the entire set of driver ABIs. These are already described in a separate document - TMG399x Device Driver v2.0.8 ABI Descriptions. Instead, this document will deal only with the ABIs used by the demo server.

The demo server devotes about 300 lines of code to ascertain, at runtime, the proper directory paths for the ABI files. For example, the path /sys/class/input/input1/als_power_state may change to …/input0/… or …/input2/… if a different number of other input devices are ever enabled before the ALS device. If your platform never varies in the number of input devices, using constant path strings is an acceptable alternative.

5.1 The gesture_data ABI

Unlike most ABIs, the gesture_data ABI does not use ASCII text strings, but uses binary data. This is done because each gesture sample, when present, requires a large amount of data. When the gesture_data ABI is opened and read, the first byte returned is a count of the number of algStatusData_t data structures that are present or 0 if no data is present. Each data structure represents 1 gesture data sample collected by the TMG399x hardware. The user program should read all of the available data at once. Up to 32 algStatusData_t data structures can be returned by each open/read/close operation. An example of reading data from the gesture_data ABI follows:

```
#define MAX_NUM_DATASETS 32
struct algStatusData valid_raw_gester_datasets[MAX_NUM_DATASETS];

int ges_raw_data_fd;
u8 count;

/* open the gesture_data ABI */
ges_raw_data_fd = open(ges_raw_data_path, O_RDONLY);
if(ges_raw_data_fd > 0)
{
    /* find out how much data is present (maybe none */
    read(ges_raw_data_fd, &count, sizeof(unsigned char));
    /* obtain any data in one read operation */
    if (count > 0)
    {
        lseek(ges_raw_data_fd, 1, SEEK_SET);
        read(ges_raw_data_fd,
             valid_raw_gester_datasets,
             (count * sizeof(struct algStatusData)));
    }
    close(ges_raw_data_fd);
}
```

5.2 The gesture_calibrate ABI

While processing gesture information, the gesture library code may determine that it is necessary to recalibrate the TMG399x offset registers. The gesture_calibrate ABI is used to trigger a new offset calibration by the kernel driver. When a calibration is required, the demo server writes a single "1" value to the gesture_calibration ABI as follows:

```
FILE *fd;

fd = fopen(ges_calibrate_path, "w");
if (fd != NULL)
{
    fprintf(fd, "%d\n", 1);
    fclose(fd);
}
```

5.3 Other ABIs

The following ABIs are read by the demo server only to display information in a log file or on the console display: prx_raw, als_red, als_green, als_blue, als_clear, als_lux, and als_cct. These ABIs are described in the TMG399x Device Driver v2.0.8 ABI Descriptions document. The demo server reads a single value from each ABI as follows:

```
i32 prox = -1;
FILE *fd;

fd = fopen(prox_raw_path, "r");
if (fd != NULL)
{
    fscanf(fd, "%d", &prox);
    fclose(fd);
}
```

6 Interfacing with Gesture Library

The gesture library is structured so that it can be implemented in a variety of architectures. In systems with enough capacity, the gesture library code can be integrated with and called directly from the kernel driver code, eliminating the need to buffer the data. The reference code samples implement the gesture library separate from the driver code and then provide a demo program to show how to pass data from the driver to the library and report the results of the library processing. The reference driver also contains an ABI which allows the demo program to create input keystroke events in response to detected gestures. This ABI is intended for demonstration purposes only and is not intended to be the final method for communicating gesture information to other applications or to the operating system.

The server plays the role of a datapump between the driver and the library. Operations that need OS services, such as file I/O (including the ABI files) or console output, should also be implemented here.

The library has one input method, Do_Raw_Gesture_Data. It expects the server to provide several output methods it can call back to, detailed in paragraph 6.2. The library is single-threaded, so a call to Do_Raw_Gesture_Data cannot return until any callback functions have returned.

6.1 Passing data to the Gesture Library

Once gesture data is obtained from the kernel driver it must be passed, one sample at a time to the gesture library code. Reading data from the gesture_driver ABI is described in paragraph 5.1 above. Once the data has been read, the user program must timestamp each sample and then pass it to the library code as follows:

```
gestureRawData_t RawData;
NSWE_t dcOffset;
u8 gexth;

for (i = 0; i < count; i++)
{
    RawData.State = valid_raw_gester_datasets[i].gRawData.state;
    RawData.NSWE = valid_raw_gester_datasets[i].gRawData.NSWE;
    RawData.GProxMax = valid_raw_gester_datasets[i].gRawData.GProx;
    RawData.Count = valid_raw_gester_datasets[i].gRawData.Count;
    clock_gettime(CLOCK_REALTIME, &(RawData.Time));
    dcOffset = valid_raw_gester_datasets[i].gesture_offset;
    gexth = valid_raw_gester_datasets[i].gesture_exit_threshold;
    Do_Raw_Gesture_Data(&RawData, &dcOffset, gexth);
}
```

6.1.1 Do_Raw_Gesture_Data

Do_Raw_Gesture_Data is the functional interface for passing data into the gesture library (as shown in paragraph 6.1 above). Call Do_Raw_Gesture_Data once for each gesture sample received.

```
void Do_Raw_Gesture_Data(gestureRawData_t *RawData,
                         NSWE_t *dcOffset,
                         u8 gexth);
```

where:

RawData contains the `adaptiveRawData_t` data from the gesture_data ABI and a timestamp added by the user,

dcOffset is copied from the gesture_data ABI,

gexth is copied from the gesture_data ABI.

6.2 Receiving data/requests from the Gesture Library

The user server program must provide 6 functions which will be called by the Gesture Library code to communicate the detection of various gesture conditions or to request that certain calibration steps be performed. Each of these functions must be implemented, even if you choose to ignore the event by simply returning from the function.

6.2.1 Do_Gesture_Long_Pushed_Event

When a gesture exceeds 300 ms, it is classified as a "button push event". When the gesture library initially detects this long event it will call the Do_Gesture_Long_Pushed_Event function. If the server program does not wish to provide button functionality, it can ignore this call simply by returning from the function.

```
void Do_Gesture_Long_Pushed_Event(RawDataArray_t *rda);
```

where:

rda is the array of gesture data for this gesture.

Do_Gesture_Long_Pushed_Event is called once for each detected long gesture.

6.2.2 Do_Gesture_Long_Held_Event

If a long gesture that has been detected and reported via the Do_Gesture_Long_Pushed_Event function continues, it will be reported by calling Do_Gesture_Long_Held_Event periodically (every 300 ms) as a "button hold event". It will continue to be reported until either the gesture ends or the gesture becomes so long that it is considered erroneous (approximately 2 seconds), in which case it will then be aborted.

```
void Do_Gesture_Long_Held_Event(RawDataArray_t *rda);
```

where:

rda is the array of gesture data for this gesture.

Do_Gesture_Long_Held_Event can be called multiple times for each detected long gesture. If the server program does not wish to provide button functionality, it can ignore this call simply by returning from the function.

6.2.3 Do_Gesture_End_Event

When the end of each normal gesture is detected, the gesture is reported by calling the Do_Gesture_End_Event function. This function is not called for long gestures (see paragraph 6.2.4 for this case).

PLEASE NOTE: In version 2.0.8 of the reference code, this function was inadvertently implemented *inside* the gesture library file named gesture_event.c. This will be corrected in the next software release and Do_Gesture_End_Event will be moved out of the library file.

```
void Do_Gesture_End_Event(RawDataArray_t *rda);
```

where:

rda is the array of gesture data for this gesture.

Do_Gesture_End_Event is called once for each detected gesture.

6.2.4 Do_Gesture_Long_Released_Event

When a long gesture ends normally (it is not aborted) it is considered to be a "button release event" and is reported by calling the Do_Gesture_Long_Released_Event function.

```
void Do_Gesture_Long_Released_Event(RawDataArray_t *rda);
```

where:

rda is the array of gesture data for this gesture.

Do_Gesture_Long_Released_Event can be called once for each detected long gesture. It will not be called if a long gesture is aborted. If the server program does not wish to provide button functionality, it can ignore this call simply by returning from the function.

6.2.5 Request_Recal_GOffset_Register

Various conditions, such as an object placed close to the sensor, or dirt/oil/makeup on the glass over the sensor can cause continuous, erroneous, gestures to be detected. When this condition occurs, the gesture library will request a recalibration operation for the gesture offset registers in the TMG399x hardware. It does this by calling the Request_Recal_GOffset_Register function.

```
void Request_Recal_GOffset_Register(void);
```

This function should pass the request to the kernel driver via the gesture_calibrate ABI as shown in paragraph 5.2.

6.2.6 Request_Visible_Data_Mode

This function has been deprecated and will be removed in the next software release. For this version, this function should simply return without performing any action.

```
void Request_Visible_Data_Mode(void);
```