LOKIICE Arduino Programming Guide 2.0 with SMART_IOT

AUTHOR

Version 2.0.0

Tue Aug 8 2023

Table of Contents

Catalog

Summary

void **connect** ()
Initiates the LOKII system with the SPI bus.

void **setCameraMode** (int cameraState)
sets LOKII camera running mode for image processing function.

int **lcdPrint** (String showString, int lineNum)
shows text on LOKII's LCD.

int **lcdPrintAll** (String showString1, String showString2, String showString3, String showString4, String showString5)
shows texts on multiple rows of LOKII's LCD.

int **waitForFaceResult** (int faceState)
waits for face detection result.

int **getFaceResult** (int attributeType)
gets attribute of the face detection result.

int **waitForBlobResult** (bool isBlocking)
waits for color blob detection result.

void **registerColor** ()
registers a custom color for color detection.

int **getBlobResult** (int blobIndex, int attributeType)
gets the attribute of the color detection result.

int **getBlobCount** ()
gets color blob detected count.

int **recordVideo** (String filename)
starts recording video from camera.

int **stopRecordVideo** ()
stops the video recording from camera.

int **playVideo** (String filename)
plays a H.264 video on the LCD screen.

int **checkVideoStatus** ()
checks video status.

int **stopPlayVideo** ()
stops the video playback on screen.

int **takePhoto** (String filename)
takes a photo.

int **displayPhoto** (String filename)
displays a jpeg photo.

void **startSpeechRecognize** (int wordgroupIndex)
starts/stops speech recognition.

int **waitForSpeechResult** ()
waits for the result of speech recognition (BLOCKING) and gets the array index of the recognized keyword.

int **getSpeechResult** ()
gets the array index of the recognized keyword (NON-BLOCKING).

int **createSDGroup** (int groupIndex, int numKeywords)
creates a speaker dependent (SD) custom speech recognition keyword groups.

int **trainSDkeyword** (int groupIndex, int keywordsIndex)
trains a speaker dependent (SD) keyword on a created keywords group.

int **checkSDComplete** (int groupIndex)
checks if speaker dependent keyword group training is completed.

void **setVolume** (int vol)
sets the sound output volume level of LOKII.

void **playTTS** (String text, int voiceType, int speed, int pitch, int emotion)
plays the english text string using LOKII built-in text-to-speech engine.

int **playSoundFile** (String filename, bool isBlocking)
plays a wav or mp3 sound file stored in the LOKII internal flash storage.

void **stopSound** ()
stops sound playback.

int **checkAudioStatus** ()
checks the audio status.

int **recAudio** (String filename)
starts audio recording.

int **stopRecAudio** (void)
stops audio recording.

int **playMIDI** (int note)
plays MIDI note.

void **setMIDIBPM** (int bpmIn)
sets MIDI BPM duration.

void **setSmartDeviceAdress** (int id)
Sets the smart device id.

int **getSmartDeviceCount** (void)
gets the number of smart devices attached in the LOKII system.

void **setMotorPower** (int motorid, int state)
Sets the smart motor power state (LOKII smart servo exclusive function).

void **moveMotor** (int motorid, int speed, int position, int direction)
Moves smart servo motor (LOKII smart servo exclusive function).

int **readMotorPosition** (int motorid)
reads smart servo motor position (LOKII smart servo exclusive function).

void **recordMotor** (int numSeconds)
records all smart servo motors' position (LOKII smart servo exclusive function).

void **playbackMotor** ()
playbacks all smart servo motors' position (LOKII smart servo exclusive function).

int **stopRecordMotor** ()
stops smart servo motors' position recording (LOKII smart servo exclusive function).

void **setRGB** (int motorid, int r, int g, int b)
sets smart LED color (LOKII smart RGB LED exclusive function).

int **setRCServo** (int motorid, int speed, int position)
moves RC servo (LOKII servo driver exclusive function).

int8_t **setRCServoCompletionTime** (int motorid, int16_t timeMS, int position)
moves RC servo to a desired position within a requested time.

int **setDCMotor** (int motorid, int speed, int direction)
moves DC motor (LOKII motor driver exclusive function).

int **getSoundDirection** ()
gets sounds direction.

int **waitForSoundDirection** ()
starts sounds direction detection.

void **startBLE** ()
initiates BLE mode.

void **resetBLE** ()
Reset the BLE device and clears all the BLE bonding information.

void **closeBLE** ()
Closes BLE connection.

int **checkBLE** ()
checks BLE status.

int **connectIOT** (int deviceCount, int *deviceIndex)
connects nearby SMART_IOT devices.

int **configIOT** (int deviceIndex, char *parameter, int paraLength)
Configures a connected BLE device.

int **commandIOT** (int deviceIndex, char *parameter, int paraLength)
sends command to a connected BLE device.

int **setSmartIOTAction** (int deviceIndex, char *parameter, int paraLength)
programmes a connected BLE device.

int **readIOTconfig** (int deviceIndex, char *parameter)
reads configuration of a BLE device.

int **readIOTSensors** (int deviceIndex, int readType, char *parameter, int *result)
reads data collected by a SMART_IOT device.

void **setWIFI** (uint8_t tar_wifiState, uint8_t tar_cloudState)
connects/disconnects WIFI and Cloud.

void **connectCloud** ()
connects Cloud.

void **disconnectCloud** ()
disconnects Cloud LOKII disconnects to LOKII-CE Cloud platform.

void **getWIFI** (uint8_t *cur_wifiState, uint8_t *cur_cloudState)
gets WIFI and Cloud connection status.

void **pushIOTToWIFI** (uint16_t times, uint16_t interval)
publish SMART_IOT sensor data to LOKII-CE Cloud.

bool **waitForQRResult** (bool isBlocking)
starts QR code detection.

void **getQRResult** (char *qrString)
get recognized QR code's string result.

void **initN55SoftSPI** (int softCSPin, int softMISOPin, int softMOSIPin, int softCLKPin)
Connects LOKII I/O Extender with SMART_Arduino.

void **setGPIO** (uint8_t ioNum, uint8_t in_or_out)
sets a LOKII I/O Extender's pin to INPUT or OUTPUT.

uint8_t **readGPIO** (uint8_t ioNum)
Reads status from a LOKII I/O Extender's pin.

void **writeGPIO** (uint8_t ioNum, uint8_t low_or_high)
Writes a LOKII I/O Extender's pin.

Function Documentation

void connect ()

Initiates the LOKII system with the SPI bus.

void setCameraMode (int cameraState)

sets LOKII camera running mode for image processing function.

sets LOKII camera to perform image processing functions.

Parameters

cameraState	defines the image processing function.
	`L_CAM_RECOGNIZE_RGB, performs Red/Green/Blue color detection mode.`
	`L_CAM_RECOGNIZE_CUSTOM, performs self-trained color detection mode.`
	`L_CAM_FACE_DETECT, performs frontal face detection mode.`
	`L_CAM_PREVIEW, sets camera to preview mode, no image processing is performed.`
	`L_CAM_V_STREAM_480X360, sets camera to video stream mode which has the resolution of 480*360.`
	`L_CAM_V_STREAM_1280X720, sets camera to video stream mode which has the resolution of 1280*720.`
	`L_CAM_V_STREAM_STOP, stops video stream mode.`
	`L_CAM_A_STREAM, sets camera to audio stream mode.`
	`L_CAM_A_STREAM_STOP, stops audio stream mode.`
	`L_CAM_QRCODE, performs QR code scan mode.`

int lcdPrint (String showString, int lineNum)

shows text on LOKII's LCD.

shows text on LOKII's LCD.

Parameters

showString	a string, maximum length is 32.
lineNum	row number between 1 - 14.

Returns

returns 1 - success, 0 - fail.

int lcdPrintAll (String showString1, String showString2, String showString3, String showString4, String showString5)

shows texts on multiple rows of LOKII's LCD.

shows texts on multiple rows of LOKII's LCD.

Parameters

showString1	a string on row 10, maximum length is 32.
showString2	a string on row 11, maximum length is 32.
showString3	a string on row 12, maximum length is 32.
showString4	a string on row 13, maximum length is 32.
showString5	a string on row 14, maximum length is 32.

Returns

returns 1 - success, 0 - fail.

int waitForFaceResult (int faceState)

waits for face detection result.

waits and returns face detection result after setCameraMode(L_CAM_FACE_DETECT) is activated. After this function call, a copy of the face result will be cached.

Parameters

faceState	defines the face detection behavior.

if faceState = L_OFF or false, this function will return immediately regardless of face detected (NON-BLOCKING).

if faceState = L_ON or true, this function will hold until a face is detected (BLOCKING).

Returns

returns face size, 0 - face not detected (in NON-BLOCKING mode only).

int getFaceResult (int attributeType)

gets attribute of the face detection result.

gets the face detection result from the cache after **waitForFaceResult(int faceState)** function is activated.

Parameters

attributeType	Following attributeTypes can be specified:
	`L_XPOS, x coordinate of the face centre.`
	`L_YPOS, y coordinate of the face centre.`
	`L_WIDTH, width of the face boundary.`
	`L_HEIGHT, height of the face centre.`
	All the above face detected atributeType are based on the LCD screen coordinate system with 320 x 240 pixel.

Returns

returns the face detected attribute result.

int waitForBlobResult (bool isBlocking)

waits for color blob detection result.

waits and returns color blob detection result after setCameraMode(L_CAM_RECOGNIZE_RGB) or setCameraMode(L_CAM_RECOGNIZE_CUSTOM) is activated. After this function call, a copy of the color result will be cached.

Parameters

isBlocking	defines the color blob detection behavior.

if isBlocking = L_OFF or false, this function will return immediately regardless of color blob detected (NON-BLOCKING).

if isBlocking = L_ON or true, this function will hold until at least one color objected is detected (BLOCKING).

Returns

returns number of color object detected, 0 - nothing detected, -1 - command sent fails.

void registerColor()

registers a custom color for color detection.

trains LOKII to recognize the registered color object. The trained custom color can then be recognized by **setCameraMode(L_CAM_RECOGNIZE_CUSTOM)** function. To have a positive training result, make sure: 1) the color object is not reflective. 2) the color object size is around 30. 3) the color object is posited at the centre of the LCD screen.

int getBlobResult (int blobIndex, int attributeType)

gets the attribute of the color detection result.

gets the color blob detected result from the cache after **waitForBlobResult(bool isBlocking)** function is activated.

Parameters

blobIndex	the index (starting from zero) for the color blob (The index should be smaller than the total color object detected).

attributeType	Following atributeTypes can be specified:
	`L_XPOS, x coordinate of the color blob centre`
	`L_YPOS, y coordinate of the color blob centre`
	`L_WIDTH, width of the color blob boundary`
	`L_HEIGHT, height of the color blob centre`
	`L_COLOR, color of the color blob, such as L_RED_COLOR, L_GREEN_COLOR, L_BLUE_COLOR, L_CUSTOM_COLOR`
	All the above color atributeTypes are based on the LCD screen coordinate system with 320 x 240 pixel

Returns

returns one of the color attributes.

int getBlobCount ()

gets color blob detected count.

gets number of color blob detected result from the cache after **waitForBlobResult(bool isBlocking)** function is activated.

Returns

returns the number of color blob in the cache.

int recordVideo (String filename)

starts recording video from camera.

Starts H.264 video recording in 480x360 resolution and saves it to internal flash storage. The maximum recording duration is 30 seconds.

Parameters

filename	string type, video name must be ended with ".avi". E.g "video.avi".

Returns

returns 1 - success, 0 - fail.

int stopRecordVideo()

stops the video recording from camera.

stops the video recording, the recorded video will be saved in LOKII internal flash storage.

Returns

returns 0 for success.

int playVideo(String filename)

plays a H.264 video on the LCD screen.

plays a H.264 video file from the LOKII internal flash storage. This function can only play H.264 video encoded by LOKII-CE boards or through the LOKII-CE video conversion tool.

Parameters

filename	the input video name with file extension ".avi". E.g "video.avi".

Returns

returns 1 for success, 0 for error.

int checkVideoStatus()

checks video status.

checks if LOKII is playing a video or not.

Returns

returns status: 1 - a video is playing, 0 - no video is playing.

int stopPlayVideo ()

stops the video playback on screen.

stops the video playback.

Returns

returns 0 for success.

int takePhoto (String filename)

takes a photo.

takes a photo with 1280x720 resolution in JPEG format and saves it in LOKII internal flash storage.

Parameters

filename	string type, the jpeg photo filename. E.g "a.jpg".

Returns

returns 1 - success, 0 - fail.

int displayPhoto (String filename)

displays a jpeg photo.

displays a jpeg photo stored in the LOKII internal flash to the LCD Screen. The maximum resolution of jpeg file is 1280x720.

Parameters

filename	the jpeg photo filename. E.g "a.jpg".

Returns

returns 1 - success, 0 - fail.

void startSpeechRecognize (int wordgroupIndex)

starts/stops speech recognition.

Performs speech recognition for trained word groups in Speaker Independent mode (SI) (index can be 1-10). Performs speech recognition for trained word groups in Speaker Dependent mode (SD) (index can be 11 - 20).

Parameters

wordgroupIndex	the index for word groups: 0 stops the speech recognition. 1 is number group for ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"]. 2 is action group for ["Action", "Move", "Turn", "Run", "Look", "Attack", "Stop", "Hello"]. 3 is movement group for ["Turn Left", "Turn Right", "Move Up", "Move Down", "Go Forward", "Go Backward"]. 4 is command group for ["Tell me a joke", "play me a song", "stop the music", "take a photo", "show me a photo", "track my face", "follow the ball", "record motor motion", "playback motor", "list commands"]. 11-20 starts speech recognition in custom training data (Speaker Dependent).

int waitForSpeechResult ()

waits for the result of speech recognition (BLOCKING) and gets the array index of the recognized keyword.

Uses with **startSpeechRecognize(int wordgroupIndex)** to get the array index of the recognized keyword when the process of speech recognition is completed, i.e Blocking.

Returns

returns an array index of the keyword or -1 for no speech recognized.

int getSpeechResult ()

gets the array index of the recognized keyword (NON-BLOCKING).

Uses with **startSpeechRecognize(int wordgroupIndex)** to get the array index of the recognized keyword immediately, i.e non-Blocking.

Returns

returns an array index of the keyword or -1 for no speech recognized.

int createSDGroup (int groupIndex, int numKeywords)

creates a speaker dependent (SD) custom speech recognition keyword groups.

creates a speech recognition with a keyword groups index (limited to 11-20). This speech recognition is speaker dependent for the training data.

Parameters

groupIndex	the index for training word groups (11-20 inclusively).
numKeywords	the number of keywords needed to train (5 maximum).

Returns

returns 1 - success , 0 - fail.

int trainSDkeyword (int groupIndex, int keywordsIndex)

trains a speaker dependent (SD) keyword on a created keywords group.

trains a speaker dependent audio on a keywords group created by **createSDGroup(int groupIndex, int numKeywords)** using LOKII built-in microphone. This speech recognition is speaker dependent for the training data.

Parameters

groupIndex	the group index of the created keywords group (11-20 inclusively).
keywordsIndex	the array index of the keyword.

Returns

returns 1 - complete , 0 - incomplete (it is recommended to train for at least 3 audio samples to get a better training result).

int checkSDComplete(int groupIndex)

checks if speaker dependent keyword group training is completed.

checks if created keyword group training is completed, i.e. all the keywords audio training is completed in the SD keyword group.

Parameters

groupIndex	the index of the created keyword group (11-20 inclusively).

Returns

returns 1 - complete (All keywords are well-trained), 0 - incomplete.

void setVolume (int vol)

sets the sound output volume level of LOKII.

sets the sound output volume level between 0-100 inclusively.

Parameters

vol	volume level (0-100).

void playTTS (String text, int voiceType, int speed, int pitch, int emotion)

plays the english text string using LOKII built-in text-to-speech engine.

converts english text string to speech with a specified voice, speed, pitch, emotion settings.

Parameters

text	english text string.

voiceType	voice can be one of following values:
	`0 = L_DEFAULT`
	`1 = L_MAN`
	`2 = L_OLDMAN`
	`3 = L_OLDWOMAN`
	`4 = L_BOY`
	`5 = L YOUNGGIRL`
	`6 = L_ROBOT`
	`7 = L_GIANT`
	`8 = L_DWARF`
	`9 = L_ALIENT`

speed	speed can be 1 - 10 inclusively.
pitch	pitch can be 1 - 10 inclusively.

emotion	emotion can be one of the following values:
	`0 = E_NATURAL`
	`1 = E_FRIENDLY`
	`2 = E_ANGRY`
	`3 = E FURIOUS`
	`4 = E_DRILL`
	`5 = E_SCARED`
	`6 = E_EMOTIONAL`
	`7 = E_WEEPY`
	`8 = E_EXCITED`
	`9 = E_SURPRISED`
	`10 = E_SAD`
	`11 = E_DISGUSTED`
	`12 = E_WHISPER`

int playSoundFile (String filename, bool isBlocking)

plays a wav or mp3 sound file stored in the LOKII internal flash storage.

plays a wav sound file by specifying the sound filename.

Parameters

filename	sound file name, e.g "a.mp3".
isBlocking	0 - non-blocking, 1 - blocking (waits until the sound file playback completed).

Returns

return status: 1 = sound file is playing, no sound file is playing.

void stopSound ()

stops sound playback.

stops sound playback.

int checkAudioStatus ()

checks the audio status.

Returns

returns the audio status: 1 = audio is playing, 0 = no audio is playing, -1 = command sent fails.

int recAudio (String filename)

starts audio recording.

starts audio recording and save it to the internal flash storage. The maximum recording duration is 30 seconds.

Parameters

filename	filename string, e.g "a.wav".

Returns

returns 0 - fail, 1 - success.

int stopRecAudio ()

stops audio recording.

Returns

returns 0 for success.

int playMIDI (int note)

plays MIDI note.

plays MIDI note for a second.

Parameters

note	the MIDI integer values (0 - 59): 0 = NOTE_C1 1 = NOTE_C1S 2 = NOTE_D1 3 = NOTE_E1b 4 = NOTE_E1 5 = NOTE_F1 6 = NOTE_F1S 7 = NOTE_G1 8 = NOTE_G1S 9 = NOTE_A2 10 = NOTE_B2b 11 = NOTE_B2 ..... If the note value is negative, LOKII will play a silence.

void setMIDIBPM (int bpmIn)

sets MIDI BPM duration.

sets MIDI BPM between 30 - 100.

Parameters

bpmIn	the BPM value.

void setSmartDeviceAdress (int id)

Sets the smart device id.

Sets the smart device id (address). In order to set the smart device id, only one smart device attached each time.

Parameters

id	the smart device id ranged between 0 - 100 inclusively.

int getSmartDeviceCount ()

gets the number of smart devices attached in the LOKII system.

Returns

returns the number of smart device.

void setMotorPower (int motorid, int state)

Sets the smart motor power state (LOKII smart servo exclusive function).

turns on/off the motor.

Parameters

motorid	the motor smart device id.
state	the power state, 0 - OFF, 1 - ON.

void moveMotor (int motorid, int speed, int position, int direction)

Moves smart servo motor (LOKII smart servo exclusive function).

Moves a specified smart servo (with 100 degree flexibility) to a desired position with a set speed.

Parameters

motorid	the servo id.
speed	the movement completion time in millsecond (100 - 2000).
position	the target moving degree (1 - 100).
direction	1 - forward, 0 - backward.

int readMotorPosition (int motorid)

reads smart servo motor position (LOKII smart servo exclusive function).

reads a specified smart servo current position.

Parameters

motorid	the servo id.

Returns

returns servo position degree.

void recordMotor (int numSeconds)

records all smart servo motors' position (LOKII smart servo exclusive function).

records all smart servo motors' position for a time duration.

Parameters

numSeconds	the duration in seconds (1- 30).

void playbackMotor ()

playbacks all smart servo motors' position (LOKII smart servo exclusive function).

playbacks smart servo motors positions which is previously recorded by **recordMotor(int numSeconds)** function.

int stopRecordMotor ()

stops smart servo motors' position recording (LOKII smart servo exclusive function).

stops smart servo motors' position recording that is triggered by **recordMotor(int numSeconds)** function.

void setRGB (int motorid, int r, int g, int b)

sets smart LED color (LOKII smart RGB LED exclusive function).

sets smart LED color using primitive Red, Green, Blue color setting.

Parameters

r	red color channel brightness (0-255).
g	green color channel brightness (0-255).
b	blue color channel brightness (0-255).

int setRCServo (int motorid, int speed, int position)

moves RC servo (LOKII servo driver exclusive function).

moves RC servo (E.g MG90S, MG996R) attached on SMART_RC board.

Parameters

motorid	the servo id.
speed	speed (0 - 20), 0 = the fastest, 1 = the slowest.
position	0 - 250 degree.

Returns

returns 0 - success, other value - fail.

int8_t setRCServoCompletionTime (int motorid, int16_t timeMS, int position)

moves RC servo to a desired position within a requested time.

moves RC servo to a desired position within a requested time.

Parameters

motorid	the servo id.
timeMS	time (ms), 16-bit integer.
position	0 - 250 degree.

Returns

returns 0 - success, other value - fail.

int setDCMotor (int motorid, int speed, int direction)

moves DC motor (LOKII motor driver exclusive function).

moves DC motor attached on SMART_POWER or SMART_DC board.

Parameters

motorid	0 - 3 for SMART_POWER connection, other id are configurable when attached on SMART_DC board.
speed	0 - 100.
direction	0 - clockwise, 1 - anti-clockwise.

int getSoundDirection ()

gets sounds direction.

gets the degree direction of sound detected from LOKII mic.

Returns

returns the degree direction of sound detected from LOKII from 1 - 180 (90 = center direction), 0 - no-reply/no-direction.

int waitForSoundDirection ()

starts sounds direction detection.

blocking = starts sounds direction detection and holds until this process is completed. Non-blocking returns the result of getSoundDirection() immediately.

Parameters

isBlocking	0 = false - non-blocking, 1 = true - blocking.

Returns

returns > 0 - the degree direction of sound detected (blocking), 0 - no-reply or LOKII set in non-blocking mode.

void startBLE ()

initiates BLE mode.

initiates BLE mode. Start BLE central mode and perform BLE scanning for any nearby SMART_IOT devices. LOKII can connect up to 8 SMART_IOT devices and stores the BLE bonding information by allocating an ID number (0-7) to a connected device at the first time. One ID for one device.

void resetBLE ()

Reset the BLE device and clears all the BLE bonding information.

Reset the BLE device and clears all the BLE bonding information. After these function call, issues **startBLE()** function will reallocate a new device ID to each paired SMART_IOT device.

void closeBLE ()

Closes BLE connection.

Turns off BLE(Bluetooth Low Energy) connection between LOKII and SMART_IOT devices.

int checkBLE ()

checks BLE status.

checks BLE(Bluetooth Low Energy) connection status of LOKII.

Returns

returns 1 - BLE off, 2 - BLE on , other values - BLE Error.

int connectIOT (int deviceCount, int *deviceIndex)

connects nearby SMART_IOT devices.

connects nearby SMART_IOT devices by specifying maximum allowed device count.

Parameters

deviceCount	number of BLE device showed. This must be greater than or equal to the input array length.
deviceIndex	output array. The caller should pass an array of character with 9 bytes memory size, as the maximum number of IOT devices connected with LOKII is 8. The passing array will show the connection status of each SMART_IOT device bonded with current SMART_SHIELD, 0 - not connected, 1 - connected. Users can check the id of the iot device and its status by printing it out. The first bytes are the status of IOT device ID 0 and the last bytes are reserved for internal use. E.g Output: {1,1,0,0,0,0,0,0,0} - SMART_IOT device ID 0 and 1 are connected.

Returns

returns number of BLE devices connected to LOKII.

int configIOT (int deviceIndex, char *parameter, int paraLength)

Configures a connected BLE device.

Writes configuration to a connected IOT device as specified by the device ID. The configuration will be clear when turning the IOT device off.

Parameters

deviceIndex	BLE device id.
parameter	array of a message. The input message contains the values of sample rate/configurations for IOT sensors: Temperature and Humidity sensor sample rate (second): parameter[0:1] - 16-bit unsigned integer or hexadecimal number into little endian. Gas sensor sampling rate (second): Fixed. Light and distance sensor sample rate (second): parameter[4:5] - 16-bit unsigned integer or hexadecimal number into little endian. 9-axis sensor sample rate (millisecond): parameter[6:7] - 16-bit unsigned integer or hexadecimal number into little endian. RGB LED light value: parameter[13:15] - 8-bit for each color channel. E.g char parameter[16] = {0,0x5,0,0,0,0x1,0xF4,0x01,0,0,0,0,0,11,13,15} parameter[0:1]: The sample rate of temperature and humidity sensor: 5 samples per second. parameter[4:5]: The sample rate of Light and distance sensor: 1 sample per second. parameter[6:7]: The sample rate of 9-axis sensor sample rate: 500 sample per millisecond. parameter[13:15]: LED RGB value: red: 11, green: 13, blue: 15.
paraLength	array length. Length must equal to 16.

Returns

returns 0 - message sent successfully, -1 - message sent failed.

int commandIOT (int deviceIndex, char *parameter, int paraLength)

sends command to a connected BLE device.

Gives command to control the peripherals of an IOT device from LOKII. Different commands can choose within parameter[0:1] by filling in 00 - buzzer or 10 - start recording sensor data in an IOT device or 20 - send recorded sensor data or 30 - LED or 40 - GPIO pins.

Parameters

deviceIndex	BLE device id from 0 - 7. The maximum number of IOT devices connected with LOKII is 8.
parameter	a message, an array char which contains commands and settings. For controlling the buzzer: parameter[0:1] = 00. parameter[2:3] - the value of interval (millisecond), 16-bit unsigned integer/hexadecimal number into little endian. parameter[4:5] - the value of frequency, 16-bit unsigned integer/hexadecimal number into little endian. To start recording data in a SMART_IOT device: parameter[0:1] = 10. parameter[2] - record time (second), 8-bit unsigned integer/hexadecimal number (the maximum value is 255). parameter[3] = 0 for internal use. **When recording, the SMART_IOT device shows "R3" on its LCD. To send data from a SMART_IOT device: parameter[0:1] = 20. **During data transmission, the SMART)_IOT device deplays "R4". It turns back to "R1", the idle status, after the work is done. User can check data by using **readIOTSensors()**. To adjust LED color: parameter[0:1] = 30. parameter[2] - red color value. parameter[3] - green color value. parameter[4] - blue color value. To turn ON/OFF I/O pin: parameter[0:1] = 40. parameter[2] - pin number, 0 or 1. **Each SMART_IOT only has two I/O pins. parameter[3] - pin status, ON = 1 and OFF = 0.
paraLength	This is the length of the message. Number shall not less than: 6 - buzzer, 4 - start recording sensor data in an IOT device, 2 - send recorded sensor data, 5 - LED, 4 - GPIO. E.g parameter[6] = {0,0,0xe8,0x03,0xd0,0x07} parameter[0:1]: 00 belongs to buzzer command. parameter[2:3]: Play buzzer every second. parameter[4:5]: Play buzzer with 2000 Hz.

Returns

returns 0 - message sent successfully, -1 - message sent failed.

int setSmartIOTAction (int deviceIndex, char *parameter, int paraLength)

programs a connected BLE device.

Sets ant action script to a connected BLE device and sends it to a SMART_IOT device.

Parameters

deviceIndex	BLE device id.
parameter	Action script string stored in the character array buffer with “null” terminated character. E.g "let te;\n\r\nsetLCDText('Temp.',0,0);\n\rte=readTemp();\n\rsetLCDText_i(te, 12, 0);\n\rif(te>40)\n\rsetLCDText('Alert',0,1);\n\r else \n\r setLCDText(' ',0,1);"
paraLength	array length.

Returns

returns 0 - message sent successfully, -1 - message sent failed.

int readIOTconfig (int deviceIndex, char *parameter)

reads configuration of a BLE device.

Reads sample rate/configuration of connected SMART_IOT device sensors in LOKII.

Parameters

deviceIndex	BLE device id.
parameter	An array of a configuration message with length = 16 bytes. The message contains the sampling rate/setting for each sensor and has the same format as shown in the example of **configIOT()**. Temperature and Humidity sensor sampling rate (second): parameter[0:1] - 16-bit unsigned integer. Gas sensor sampling rate (second): Fixed. Light and distance sensor sampling rate (second): parameter[4:5] - 16-bit unsigned integer. 9-axis sensor sampling rate (milliseconds): parameter[6:7] - 16-bit unsigned integer. RGB LED light value: parameter[13:15] - 8 bit LED color for RED/GREEN/BLUE channel. *All values are in little endian format.
paraLength	array length. Array's size shall not less than 16. E.g The following array was printed in decimal values: char parameter[16] = {5,0,0,0,1,0,244,1,0,0,0,0,0,11,13,15} parameter[0:1]: The sample rate of temperature and humidity sensor = 0x05 = 5 samples per second. parameter[4:5]: The sample rate of Light and distance sensor = 0x01 = 1 sample per second. parameter[6:7]: The sample rate of 9-axis sensor sample rate = 0x01F4 = 500 sample per millisecond. parameter[13:15]: LED RGB value: red: 11, green: 13, blue: 15.

Returns

returns 0 - message sent successfully, -1 - message sent failed.

int readIOTSensors (int deviceIndex, int readType, char * parameter, int * result)

reads data collected by a SMART_IOT device.

reads data collected by a connected SMART_IOT device on LOKII.

Parameters

deviceIndex	BLE device id.
readType	0 - read current sensor data, 1 - read recorded "sensor data" header, 2 - read recorded "sensor data" content.
parameter	array address of a message. Data will pass into the message. See Arduino_BLE_IOTcommandRecordData.ino.
result	int buffer address to store the collected data. result = 0 means no recorded sensor data.

Returns

returns 0 - message sent successfully, -1 - message sent failed.

void setWIFI (uint8_t tar_wifiState, uint8_t tar_cloudState)

connects/disconnects WIFI and Cloud.

LOKII connects/disconnects to WIFI or Cloud (Non-blocking).

Parameters

tar_wifiState	1 - connects to WIFI, 0 - disconnects to WIFI.
tar_cloudState	1 - connects to Cloud, 0 - disconnects to Cloud. Default value should always set to zero.

void connectCloud ()

connects Cloud.

LOKII connects to LOKII-CE Cloud platform for SMART_IOT data uploading and control (Blocking). This function requires monthly subscription service. Please send request to www.btobsteam.com for pricing issue.

void disconnectCloud ()

disconnects Cloud LOKII disconnects to LOKII-CE Cloud platform.

void getWIFI (uint8_t * cur_wifiState, uint8_t * cur_cloudState)

gets WIFI and Cloud connection status.

gets WIFI and Cloud connection status from LOKII.

Parameters

cur_wifiState	0 - disconnection, 1 - connection.
cur_ckoudState	0 - disconnection, 1 - connection.

void pushIOTToWIFI (uint16_t times, uint16_t interval)

publish SMART_IOT sensor data to LOKII-CE Cloud.

sends current sensor data to Cloud from a connected SMART_IOT device in a number of times.

Parameters

times	a number of times to upload, unlimited times = 0.
interval	Interval between each SMART_IOT sensor data uploading to cloud. E.g 60 - upload data for every minute. 0 - stop upload.

bool waitForQRResult (bool isBlocking)

starts QR code detection.

blocking = starts QR code detection and waits until QR code is recognized. Non-blocking returns the result immediately from current live camera image.

Parameters

isBlocking	0 = false - non-blocking, 1 = true - blocking.

Returns

returns 1 - success, 0 - no QR code scanned.

void getQRResult (char *qrString)

gets recognized QR code's string result.

Get QR code's string result. Caller should allocate at least 128 bytes character buffer for the qrString variable in order to get the QR result.

Parameters

qrString	char array pointer. The recognized result will be passed into this memory address.

void initN55SoftSPI (int softCSPin, int softMISOPin, int softMOSIPin, int softCLKPin)

Connects LOKII I/O Extender with SMART_Arduino.

a method initiates a communication bridge between SMART_ARDUINO and LOKII I/O Extender.

Parameters

softCSPin	the corresponding SMART_Arduino's pin connects to SPI_CS of I/O Extender.
softMISOPin	the corresponding SMART_Arduino's pin connects to SPI_MISO of I/O Extender.
softMOSIPin	the corresponding SMART_Arduino's pin connects to SPI_MOSI of I/O Extender.
softCLKPin	the corresponding SMART_Arduino's pin connects to SPI_SCK of I/O Extender.

void setGPIO(uint8_t ioNum, uint8_t in_or_out)

sets a LOKII I/O Extender's pin to INPUT or OUTPUT.

a method sets a LOKII I/O Extender's pin to INPUT or OUTPUT.

Parameters

ioNum	a pin number of I/O Extender from 0 to 16.
in_or_out	0, input pull-up. 1, output pull-down.

uint8_t readGPIO (uint8_t ioNum)

Reads status from a LOKII I/O Extender's pin.

a method reads status from a LOKII I/O Extender's pin.

Parameters

ioNum	a pin number of I/O Extender from 0 to 16.

Returns

returns status of a pin.

void writeGPIO (uint8_t ioNum, uint8_t low_or_high)

Writes a LOKII I/O Extender's pin.

a method writes a LOKII I/O Extender's pin.

Parameters

ioNum	a pin number of I/O Extender from 0 to 16.
low_or_high	0 = low. 1 = high.

Example Documentation

Clamp_and_Arm_with_Object_Demo.ino

This example shows the usage of **setRCServo()** functions to control RC servos using SMART_RC board (J1,J2,J3,J4 port). The program moves the arm to the right hand side and uses the claw to pick up an object. Then, the arm will move to the other side and place the object. Before running the program, please make sure the LOKII arm and claw RC servos are connected to the right RC servo ports. i.e. SMART_SERVO: J1 = ID 8, J2 = ID 9, J3 = ID 10, J4 = ID 11. In this case, the servo (J4) controls the claw , the servo (J3) controls the arm moving left and right, the servo (J2) controls the arm moving up and down and the servo (J1) controls the arm moving forward and backward respectively.

```
#include <stdio.h>
#include <lokii.h>
LOKII lokii;

int const Turnable_Mounting = 10;
int const Arm_Up_and_Down = 9;
int const Arm_Backward_and_Forward = 8;
int const Clamp = 11;
int times = 0;

void Normal()
{
  lokii.setRCServo(Clamp,5,99);
  delay(1000);
  lokii.setRCServo(Arm_Backward_and_Forward,5,169);
  delay(1000);
  lokii.setRCServo(Arm_Up_and_Down,5,5);
  delay(1000);
  lokii.setRCServo(Turnable_Mounting,5,125);
  delay(1000);
}

void setup() {
  // put your setup code here, to run once:
  while(!Serial);
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  Normal();
}

void loop() {
  // put your main code here, to run repeatedly:
  if (times == 0)
  {
    lokii.setRCServo(Arm_Up_and_Down,5,1);
    delay(1000);
    lokii.setRCServo(Turnable_Mounting,5,200);
    delay(1000);
    lokii.setRCServo(Arm_Up_and_Down,5,70);
    delay(1000);
    lokii.setRCServo(Arm_Backward_and_Forward,5,200);
    delay(1000);
    lokii.setRCServo(Clamp,5,40);
    delay(1000);
    lokii.setRCServo(Arm_Backward_and_Forward,5,169);
    delay(1000);
    lokii.setRCServo(Arm_Up_and_Down,5,1);
    delay(1000);
    lokii.setRCServo(Turnable_Mounting,5,50);
    delay(1000);
    lokii.setRCServo(Arm_Up_and_Down,5,70);
    delay(1000);
```

```
    lokii.setRCServo(Arm_Backward_and_Forward,5,200);
    delay(1000);
    Normal();
    times = 1;
  }
}
```

Color_Detection.ino

This example setups LOKII into RGB color detection mode and retrieves the color object and its attributes detected by waitForBlobResult(1) in the blocking mode.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int PosX, posY, width, height, color;

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_RECOGNIZE_RGB);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  lokii.waitForBlobResult(true);
  PosX = lokii.getBlobResult(0, L_XPOS);
  posY = lokii.getBlobResult(0, L_YPOS);
  width = lokii.getBlobResult(0, L_WIDTH);
  height = lokii.getBlobResult(0, L_HEIGHT);
  color = lokii.getBlobResult(0, L_COLOR);
  Serial.println(PosX); //check output on Serial
}
```

DC_Motor.ino

This examples move 4 DC motors in a clockwise and anti-clockwise direction using 4 DC motor ports on the SMART_POWER boards.DC motors with id 0 and 1 are connected in the DC motor port near the SMART_BUS port on the SMART_POWER. On the other hand, DC motors with id 1 and 2 are connected in the DC motor port on the other sides of SMART_POWER board.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  Serial.begin(9600);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //run only once
  {
    times = times + 1;
    lokii.playTTS("Go forward.", 0, 5, 5, 0);
    for(int i = 0; i < 15; i++)
    {
      lokii.setDCMotor(0,26,0); //forward
      lokii.setDCMotor(1,26,0);
      lokii.setDCMotor(2,26,0);
      lokii.setDCMotor(3,26,0);
    }
    lokii.playTTS("Turn left.", 0, 5, 5, 0);
    for(int i = 0; i < 3; i++)
    {
      lokii.setDCMotor(0,26,0); //left
      lokii.setDCMotor(1,26,0);
      lokii.setDCMotor(2,0,0);
      lokii.setDCMotor(3,0,0);
    }
    lokii.playTTS("Turn Right.", 0, 5, 5, 0);
    for(int i = 0; i < 3; i++)
    {
      lokii.setDCMotor(0,0,0); //right
      lokii.setDCMotor(1,0,0);
      lokii.setDCMotor(2,26,0);
      lokii.setDCMotor(3,26,0);
    }
    lokii.playTTS("Go backward.", 0, 5, 5, 0);
    for(int i = 0; i < 15; i++)
    {
      lokii.setDCMotor(0,26,1); //backward
      lokii.setDCMotor(1,26,1);
      lokii.setDCMotor(2,26,1);
      lokii.setDCMotor(3,26,1);
    }
  }
  lokii.setDCMotor(0,0,0);
  lokii.setDCMotor(1,0,0);
  lokii.setDCMotor(2,0,0);
  lokii.setDCMotor(3,0,0);
}
```

Face_Detection.ino

This example setups LOKII into Face detection mode and retrieves the biggest human front face and its attributes detected by using waitForFaceResult(1) in the blocking mode.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int faceResult, PosX, posY, width, height;

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_FACE_DETECT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  lokii.waitForFaceResult(1);
  PosX = lokii.getFaceResult(L_XPOS);
  posY = lokii.getFaceResult(L_YPOS);
  width = lokii.getFaceResult(L_WIDTH);
  height = lokii.getFaceResult(L_HEIGHT);
  Serial.println(PosX); //check output on Serial
}
```

Keyword_Group_Training.ino

This example only trains five new keywords for three times and registered the trained keywords group at index 11. After the trained keywords is successfully created, LOKII will recognized the keyword and play the keyword the user spoke if succeed. This new keyword group 11 also contains number group's keywords, so the index of new keywords starts from 11. Keyword group 11 will look like List[] below.

```
#include <lokii.h>

volatile int SD_index;
volatile int numKeywords;
volatile int result;

LOKII lokii;


String numberList[] = {"orange", "apple", "mango", "banana", "grape"};
String List[] = {"zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine",
"ten", "orange", "apple", "mango", "banana", "grape"};


void setup(){
  Serial.begin(9600);

  lokii.connect();

  SD_index = 11;
  numKeywords = 5;
  result = 0;
}

void loop(){

  int speechResult;

  // create speaker depentent group
  lokii.createSDGroup(SD_index, numKeywords);
  for (int i = (0); i <= (numKeywords - 1); i = i + (1)) {

      lokii.playTTS(  (String) "Said Your word" + (String) " " , L_DEFAULT ,5 ,5 ,E_NATURAL);
      Serial.println(String("Train speech index") + String(String(i, DEC)));
      //delay(500);
      result = lokii.trainSDkeyword(SD_index,i);
      Serial.println(String(" result = ") + String(String(result, DEC)));
  }
  result = lokii.checkSDComplete(SD_index);
  if (result == 0) {
    Serial.println("Create SD group success!");

  } else {
    Serial.println("Create SD group Fail!");
  }


  // start recognize speaker depentent group
  while (1)
  {
      lokii.startSpeechRecognize(SD_index);
      speechResult = lokii.waitForSpeechResult();
      if(speechResult>=0 && speechResult <= (10+numKeywords))
      {
        lokii.playTTS(List[speechResult] , L_DEFAULT ,5 ,5 ,E_NATURAL);
      }
  }
```

playmidi.ino

Different MIDI note numbers are assigned to the playMIDI function to play "Twinkle Twinkle Little Star", "It's a Small World", "Open Mos" with a frequency set by setMIDIBPM. (Beat Per Minutes)

```
/*
  songnote

  [Prerequisite]
  This program requires a speaker attached to LOKII Arduino Shield.

  [Function]
  Set a smart ID to the serial device attached in LOKII Arduino Shield.
  (Make sure only one smart device is attached)
  User shall enter a smart ID number [ 1-99] to the serial monitor connected in Arduino
  when prompt to set the smart ID.


  Written by Markuz Lee
  10-1-2020

*/


#include <lokii.h>

LOKII lokii;

int TwinkleLittleStar[] =
{
  BEAT_1, NOTE_1, NOTE_1, NOTE_5, NOTE_5, NOTE_6, NOTE_6, BEAT_2, NOTE_5,
  BEAT_1, NOTE_4, NOTE_4, NOTE_3, NOTE_3, NOTE_2, NOTE_2, BEAT_2, NOTE_1,
  BEAT_1, NOTE_5, NOTE_5, NOTE_4, NOTE_4, NOTE_3, NOTE_3, BEAT_2, NOTE_2,
  BEAT_1, NOTE_5, NOTE_5, NOTE_4, NOTE_4, NOTE_3, NOTE_3, BEAT_2, NOTE_2,
  BEAT_1, NOTE_1, NOTE_1, NOTE_5, NOTE_5, NOTE_6, NOTE_6, BEAT_2, NOTE_5,
  BEAT_1, NOTE_4, NOTE_4, NOTE_3, NOTE_3, NOTE_2, NOTE_2, BEAT_2, NOTE_1,
};




int openMos[] =
{ BEAT_1, NOTE_1, NOTE_2, NOTE_3, NOTE_1, NOTE_1, NOTE_2, NOTE_3, NOTE_1, NOTE_3, NOTE_4, BEAT_2,
NOTE_5,
  BEAT_1, NOTE_3, NOTE_4, BEAT_2, NOTE_5,
  BEAT_0_5, NOTE_5, NOTE_6, NOTE_5, NOTE_4, BEAT_1, NOTE_3, NOTE_1,
  BEAT_0_5, NOTE_5, NOTE_6, NOTE_5, NOTE_4, BEAT_1, NOTE_3, NOTE_1,
  BEAT_1, NOTE_2, NOTE_5L, BEAT_2, NOTE_1,
  BEAT_1, NOTE_2, NOTE_5L, BEAT_2, NOTE_1,
};


int smallWorld[] =
{

  BEAT_1, NOTE_3L, NOTE_4L, BEAT_2, NOTE_5L, NOTE_3, NOTE_1,
  BEAT_1, NOTE_2, NOTE_1, BEAT_2, NOTE_1, NOTE_7L, NOTE_7L,
  BEAT_1, NOTE_2L, NOTE_3L, BEAT_2, NOTE_4L, NOTE_2, NOTE_7L,
  BEAT_1, NOTE_1, NOTE_7L, BEAT_2, NOTE_6L, NOTE_5L, NOTE_5L,
  BEAT_1, NOTE_3L, NOTE_4L, BEAT_2, NOTE_5L, BEAT_1, NOTE_1, NOTE_2, BEAT_2, NOTE_3,
  BEAT_1, NOTE_2, NOTE_1, BEAT_2, NOTE_6L, BEAT_1, NOTE_2, NOTE_3, BEAT_2, NOTE_4,
  BEAT_1, NOTE_3, NOTE_2, BEAT_2, NOTE_5L, NOTE_4, NOTE_3, NOTE_2, BEAT_4, NOTE_1,

  BEAT_4, NOTE_P,

  BEAT_1, NOTE_1, NOTE_P, NOTE_P, NOTE_1, BEAT_2, NOTE_3, NOTE_1, NOTE_2, BEAT_1, NOTE_P, NOTE_2,
BEAT_4, NOTE_2,
```

```
  BEAT_1, NOTE_2, NOTE_P, NOTE_P, NOTE_2, BEAT_2, NOTE_4, NOTE_2, NOTE_3, BEAT_1, NOTE_P, NOTE_3,
BEAT_4, NOTE_3,
  BEAT_1, NOTE_3, NOTE_P, NOTE_P, NOTE_3, BEAT_2, NOTE_5, NOTE_3, NOTE_4, BEAT_1, NOTE_P, NOTE_4,
BEAT_2, NOTE_4,
  BEAT_1, NOTE_3, NOTE_2, BEAT_4, NOTE_5L, NOTE_7L, NOTE_1

} ;



// 1 = C // Do // C4?
// 2 = D // Re
// 3 = E // Mi
// 4 = F
// 5 = G
// 6 = A
// 7 = B


void setup() {

  Serial.begin(9600);
  //while (!Serial);

  lokii.connect();
  Serial.println("starting E!");
  lokii.setVolume(100);

}

void loop() {

  // beat per minute
  //lokii.setMIDIBPM(90);

  // play single note Middle Major C for 1 beat
  // lokii.playMIDI(BEAT_1);
  // lokii.playMIDI(NOTE_C);


  // set LOKII to 90 beat per minute
  lokii.setMIDIBPM(90);

  // play "Twinkle Twinkle Little Star"
  for (unsigned int i = (1); i <= (sizeof(TwinkleLittleStar) / sizeof(TwinkleLittleStar[0]));
i = i + (1)) {
    lokii.playMIDI(TwinkleLittleStar[(int)(i - 1)]);
  }


  // set LOKII to 300 beat per minute
  lokii.setMIDIBPM(300);
  // play "It's a Small World"
  for (int i = (1); i <= (sizeof(smallWorld)/sizeof(smallWorld[0])); i = i + (1)) {
    lokii.playMIDI(smallWorld[(int)(i - 1)]);
  }


  // set LOKII to 100 beat per minute
  lokii.setMIDIBPM(100);
  // play "Open Mos"
  for (int i = (1); i <= (sizeof( openMos   )/sizeof(openMos[0])); i = i + (1)) {
     lokii.playMIDI( openMos[(int)(i - 1)]);
  }

  delay(10000);

}
```

recordAudio.ino

An audio will be recorded for 10 seconds and saved to a file called "b.wav" in LOKII internal flash. Then, the recorded sound file will be playback by LOKII.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while(!Serial);
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
    Serial.println("test");
    Serial.println(lokii.checkAudioStatus());
    lokii.playTTS("Record an Audio", 0, 5, 5, 0);
    lokii.recAudio("b.wav");
    delay(10000);
    lokii.stopRecAudio();
    lokii.playTTS("audio saved", 0, 5, 5, 0);
    delay(1000);
    lokii.playSoundFile("b.wav",1);
    delay(10000);
    lokii.stopSound();
  }
}
```

recordVideo.ino

This program will take a picture and record a video for 5 seconds and the files will be stored in LOKII internal flash. Then, it will playback the video and display a photo.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
    lokii.playTTS("Take a photo", 0, 5, 5, 0);
    lokii.takePhoto("a.jpg");
    lokii.recordVideo("video6.avi");
    delay(5000);
    lokii.stopRecordVideo();
    lokii.playTTS("Video saved", 0, 5, 5, 0);
    delay(3000);
    lokii.playVideo("video6.avi");
    delay(5000);
    lokii.stopPlayVideo();
    lokii.displayPhoto("a.jpg");
  }
}
```

playVideo.ino

This program will play three different videos subsequently.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
String videoList[3] = {"1280_10MG.mov","car.mov","O1.mov"};
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
    for(int i = 0; i < sizeof(videoList); i++)
    {
      lokii.playVideo(videoList[i]);
      while(lokii.checkVideoStatus())
      {
        Serial.println("video is playing");
      }
    }
  }
}
```

SoundDirection_blocking.ino

LOKII detects the sound direction and print out the sound directed detected.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
   int i = lokii.waitForSoundDirection(1);
   Serial.println(i);
}
```

Speech_Recognition.ino

LOKII recognizes the keywords from built-in keywords group 4 and speak out the recognized keywords using Text-To-Speech functions.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;

int speechResult;
String command_group[10] = {"Tell me a joke", "play me a song", "stop the music", "take a photo",
"show me a photo", "track my face", "follow the ball", "record motor motion", "playback motor",
"list commands"};

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
}

void loop() {
  // put your main code here, to run repeatedly:
  lokii.startSpeechRecognize(4);
  speechResult = lokii.waitForSpeechResult();
  if(speechResult >= 0 && speechResult <=9)
  {
    lokii.startSpeechRecognize(0);
    lokii.playTTS(command_group[speechResult],0,5,5,0);
  }
}
```

TTS.ino

LOKII deploys Text-To-Speech function to speak out different English text using a combination of settings of voice types, speed, pitch and emotion types.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
   lokii.playTTS("Twinkle, twinkle, little star,", L_DEFAULT, 5, 5, E_NATURAL);
   lokii.playTTS("How I wonder what you are!", L_DEFAULT, 5, 5, E_NATURAL);
   lokii.playTTS("Up above the world so high,", L_DEFAULT, 5, 5, E_NATURAL);
   lokii.playTTS("Like a diamond in the sky", L_DEFAULT, 5, 5, E_NATURAL);
   lokii.playTTS("Twinkle, twinkle, little star,", L_YOUNGGIRL, 5, 5, E_FRIENDLY);
   lokii.playTTS("How I wonder what you are!", L_MAN, 5, 5, E_EMOTIONAL);
   lokii.playTTS("Up above the world so high,", L_BOY, 5, 5, E_EXCITED);
   lokii.playTTS("Like a diamond in the sky", L_OLDWOMAN, 5, 5, E_SURPRISED);
  }
}
```

Arduino_BLE.ino

This example shows how to connect BLE device and display the ID of the connected SMART_IOT. The maximum number of SMART_IOT device LOKII can connect is 8 and ID starts from 0 to 7. The last element of the array remains for internal use.

```
#include <lokii.h>


int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int cnt = 1;


LOKII lokii;

void setup() {
  Serial.begin(9600);
  while(!Serial)
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();
  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    for(int i = 0; i < (MAX BLE DEVICES+1); i++)
    {
      Serial.print("ID");
      Serial.print(i);
      Serial.print(":");
      Serial.println(connectedBLE[i]);
      Serial.println("connected");
    }
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));
    //cnt++;
    Serial.println("abc1");
  }
  cnt++;
  if(cnt == 30)
  {
    cnt = 1;
  }
}
```

Arduino_BLE_IOTconfigurations.ino

To configure SMART_IOT devices, LOKII connects the devices and write their configuration through sending the array of the message. This example shows one of the two ways on how to create the message. Users can use either struct or array for that.

```
#include <lokii.h>


int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int list[MAX_BLE_DEVICES];
int cnt = 1;
char param[16];
int result;

typedef struct
{
  uint16_t temperature_and_humidity_interval_second; // temperature and humidity sample time,
unit : second
  uint16_t NC1;
  uint16_t light_interval_second; // proximity sample time, unit : second
  uint16_t imu_interval_ms; // 9-axis data sample time, unit : ms
  uint8_t  NC19;
  uint8_t  NC20;
  uint8_t  NC21;
  uint8_t  NC30;
  uint8_t  NC31;
  uint8_t  led_red;
  uint8_t  led_green;
  uint8_t  led_blue;
} IOTConfig_t;
IOTConfig_t deviceConfig[MAX_BLE_DEVICES];

LOKII lokii;
void setup() {
  param[0] = 0; //uint16_t temperature_and_humidity_interval_second(s)
  param[1] = 0x5; //temperature_and_humidity_interval_second(s)
  param[2] = 0;
  param[3] = 0;
  param[4] = 0; //uint16_t light_interval_second(s)
  param[5] = 0x1; //light_interval_second(s)
  param[6] = 0xF4; //uint16_t 9-axis sensor sample rate(ms)
  param[7] = 0x01; //9-axis sensor sample rate(ms)
  param[8] = 0; //uint8_t 9-axis sensor configurations
  param[9] = 0;
  param[10] = 0;
  param[11] = 0;
  param[12] = 0;
  param[13] = 0x11; //uint8_t led_red
  param[14] = 0x13; //uint8_t led_green
  param[15] = 0x15; //uint8_t led_blue


  Serial.begin(9600);
  while(!Serial)
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();

  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));
    for(int i = 0; i < (MAX_BLE_DEVICES); i++)
    {
```

```
    if(connectedBLE[i] == 1)
    {
      Serial.print("ID");
      Serial.print(i);
      Serial.print(" is connected.");

      deviceConfig[i].temperature_and_humidity_interval_second = 5;
      deviceConfig[i].light_interval_second = 1;
      deviceConfig[i].imu_interval_ms = 500;
      deviceConfig[i].led_red = 11;
      deviceConfig[i].led_green = 13;
      deviceConfig[i].led_blue = 15;

      //int abc = lokii.configIOT(i, (char*) & (deviceConfig[i]), sizeof(IOTConfig_t));
      int abc = lokii.configIOT(i,param,sizeof(param));
      Serial.println(abc);
      if(abc == LOKI_REPLY_OK)
      {
        Serial.print("ID");
        Serial.print(i);
        Serial.print(" is configured!");
      }

    }
  }
  for (int i = 0; i < MAX_BLE_DEVICES; i++)
  {
    if (connectedBLE[i] != 0)
    {
      if (lokii.readIOTconfig(i,param) == LOKI REPLY OK)
      {
         Serial.print(F("read device "));
         Serial.print(i);
         Serial.println(F(" 's config success"));
         for(int j = 0; j < sizeof(param); j++)
         {
          Serial.println(int(param[j]));
         }
      }
      else
      {
        Serial.print(F("read device "));
        Serial.print(i);
        Serial.println(F(" 's config fail"));
      }
    }
  }
  }
  cnt++;
}
```

Arduino_BLE_IOTcommandPlayingBuzzer.ino

LOKII gives a command to connected SMART_IOTs to play the Buzzer. The buzzer command is "00" and the command in this example will set the buzzer to play with 2000 Hz every second.

```
#include <lokii.h>


int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int cnt = 1;
char param[256];
int result;


LOKII lokii;

void setup() {

  param[0] = 0;
  param[1] = 0;
  param[2] = 0xe8;
  param[3] = 03;
  param[4] = 0xd0;
  param[5] = 07;


  Serial.begin(9600);
  while(!Serial)
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();

  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));
    for(int i = 0; i < (MAX_BLE_DEVICES); i++)
    {
      if(connectedBLE[i] == 1)
      {
        Serial.print("ID");
        Serial.print(i);
        Serial.print(" is connected.");
        Serial.println(lokii.commandIOT(i,param,6));

      }
    }
    cnt++;
  }
}
```

Arduino_BLE_IOTcommandRecordData.ino

LOKII commands the connected SMART_IOTs to record data for 60 seconds with a command "10" and collect the recorded data back from the devices with using a command "20". To read the recorded data, use **readIOTSensors()** to read whether the data header (data size, recording time) or the data. Users can also get current sensor data by setting the second input argument as "0".

```
#include <lokii.h>

int ID;
int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int cnt = 1;
char param[256];
char param1[256];
int result;
uint32_t dataRecordSize;
uint32_t dataRecordStartTime;
uint32_t dataRecordEndTime;
uint16_t sensorType;
uint16_t sensorData16[3];


LOKII lokii;

void printIMUSensorData(int16_t aX, int16_t aY, int16_t aZ,
                    int16_t gX, int16_t gY, int16_t gZ,
                    int16_t mX, int16_t mY, int16_t mZ)
{

      Serial.print(F("accel: "));
      Serial.print(0.000488 * aX);
      Serial.print(F(" G "));
      Serial.print(0.000488 * aY);
      Serial.print(F(" G "));
      Serial.print(0.000488 * aZ);
      Serial.println(F(" G "));

      //sensitivity = 70 mdps/LSB , +-2000 degree/second
      Serial.print(F("gyro: "));
      Serial.print(0.07 * gX);
      Serial.print(F(" Deg/s "));
      Serial.print(0.07 * gY);
      Serial.print(F(" Deg/s "));
      Serial.print(0.07 * gZ);
      Serial.println(F(" Deg/s "));

      // sensitivity 1/1711 gauss/LSB   , +-1600
      // 1 guass = 100 microTesla
      Serial.print(F("mag: "));
      Serial.print((float)100 / 1711 * mX);
      Serial.print(F(" uT "));
      Serial.print((float)100 / 1711 * mY);
      Serial.print(F(" uT "));
      Serial.print((float)100 / 1711 * mZ);
      Serial.println(F(" uT "));
}

void setup() {
  //start recording sensor data
  param[0] = 1;
  param[1] = 0;
  param[2] = 0x3c;//60 sec
  param[3] = 0;
  //send recorded data
  param1[0] = 2;
  param1[1] = 0;
  Serial.begin(9600);
  while(!Serial)
```

```
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();

  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));
    for(int i = 0; i < (MAX_BLE_DEVICES); i++)
    {
      if(connectedBLE[i] == 1)
      {
        Serial.print("ID");
        Serial.print(i);
        Serial.print(" is connected.");
        ID = i;
      }
    }
    cnt++;
    Serial.println(lokii.commandIOT(ID,param,6));//record data
    delay(60000);
    lokii.commandIOT(ID,param1,2);//send data
    if(lokii.readIOTSensors(ID, 1, param, &result) == LOKI_REPLY_OK) //LOKI_REPLY_OK = 0, a
constant stored in LOKII library. //1 = read recorded "sensor data" header
    {
      if(result == 0)
      {
        Serial.println("no recorded data");
        return;
      }

      dataRecordSize =     param[0] + (param[1] << 8) + (param[2] << 16) + (param[3] << 24);
      dataRecordStartTime = param[4] + (param[5] << 8) + (param[6] << 16) + (param[7] << 24);
      dataRecordEndTime =   param[8] + (param[9] << 8) + (param[10] << 16) + (param[11] << 24);
      Serial.print(F("Record size: "));
      Serial.print(dataRecordSize);
      Serial.print(F(" StartTime: "));
      Serial.print(dataRecordStartTime);
      Serial.print(F(" EndTime "));
      Serial.println(dataRecordEndTime);

      while (lokii.readIOTSensors(ID, 2, param, &result) == LOKI_REPLY_OK && result > 0) //
2 = read recorded "sensor data" content
      {

        // structure : Lenght(1byte , total length of timestamp + sensor data type + sensor data)
+ timestamp tick(4byte) + sensor data type(2byte)+  sensor data

       // param[0] = length of this single record data, should be equal to result-1;

        dataRecordStartTime = param[1] + (param[2] << 8) + (param[3] << 16) + (param[4] << 24);
        sensorType = param[5] + (param[6] << 8);

        if (sensorType == 1) // temperature
        {
           Serial.print("Record Time: ");
           Serial.print(dataRecordStartTime);
           Serial.print(" temperature: ");
           Serial.print((int)param[7]);
           Serial.print(".");
           Serial.print((int)param[8]);
           Serial.println(" Degree");
        }
        else if (sensorType == 2) // humidity
        {
           Serial.print("Record Time: ");
           Serial.print(dataRecordStartTime);
           Serial.print(" humidity: ");
```

```
        Serial.println((int)param[7]);
      }
      else if (sensorType == 3) // tvoc
      {
        sensorData16[1] = param[9] + (param[10] << 8);
        Serial.print("Record Time: ");
        Serial.print(dataRecordStartTime);
        Serial.print(" tvoc: ");
        Serial.println(sensorData16[1]);
      }
      else if (sensorType == 4) // light
      {
        sensorData16[0] = param[7] + (param[8] << 8);
        sensorData16[1] = param[10] + (param[11] << 8);
        Serial.print("Record Time: ");
        Serial.print(dataRecordStartTime);
        Serial.print(" lux: ");
        Serial.print(sensorData16[0]);
        Serial.print(" distance: ");
        Serial.print((int)param[9]);
        Serial.print(" mm soundLevel: ");
        Serial.print(sensorData16[1]);
        Serial.println(" DB");
      }
      else if (sensorType == 7)
      {
        // imu sensor 16bit raw data
        Serial.print(F("Record Time: "));
        Serial.println(dataRecordStartTime);

        int16_t aX, aY, aZ;
        aX = param[7] + (param[8] << 8);
        aY = param[9] + (param[10] << 8);
        aZ = param[11] + (param[12] << 8);
        int16_t gX, gY, gZ;
        gX = param[13] + (param[14] << 8);
        gY = param[15] + (param[16] << 8);
        gZ = param[17] + (param[18] << 8);
        int16_t mX, mY, mZ;
        mX = param[19] + (param[20] << 8);
        mY = param[21] + (param[22] << 8);
        mZ = param[23] + (param[24] << 8);

        printIMUSensorData(aX, aY, aZ, gX, gY, gZ, mX, mY, mZ);
      }
    }
   }
   else
   {
     Serial.println(F("read recorded sensor header: fail"));
   }
  }
}
```

Arduino_BLE_IOTcommandLEDon_off.ino

A SMART_IOT LED can be changed to different color by giving a command "30" from LOKII. The last three elements of the message are the color values of RGB.

```
#include <lokii.h>


int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int cnt = 1;
char param[256];
int result;


LOKII lokii;

void setup() {

  param[0] = 3;
  param[1] = 0;
  param[2] = 0xff;
  param[3] = 0xff;
  param[4] = 0xff;


  Serial.begin(9600);
  while(!Serial)
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();

  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));
    for(int i = 0; i < (MAX_BLE_DEVICES); i++)
    {
      if(connectedBLE[i] == 1)
      {
        Serial.print("ID");
        Serial.print(i);
        Serial.print(" is connected.");
        Serial.println(lokii.commandIOT(i,param,5));

      }
    }
    cnt++;
  }
}
```

Arduino_BLE_IOTcommandRelay.ino

User can turn I/O pins of SMART_IOTs ON/OFF by giving a command "40" from LOKII. There are two I/O pins on each SMART_IOT which are numbered as "0" and "1".

```
#include <lokii.h>


int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int cnt = 1;
char param[256];
int result;


LOKII lokii;

void setup() {

  param[0] = 4;
  param[1] = 0;
  param[2] = 1; // pin 1
  param[3] = 1;// on


  Serial.begin(9600);
  while(!Serial)
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();

  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));
    for(int i = 0; i < (MAX_BLE_DEVICES); i++)
    {
      if(connectedBLE[i] == 1)
      {
        Serial.print("ID");
        Serial.print(i);
        Serial.print(" is connected.");
        Serial.println(lokii.commandIOT(i,param,4));

      }
    }
    cnt++;
  }
}
```

Arduino_BLE_setIOTaction.ino

This example shows how LOKII programs the SMART_IOTs by delivering the script via BLE. The script is based on a programming language called ELK (Extension Language Kit). For more information of the script, please refer to the document for SMART_IOT.

```
#include <lokii.h>


int const MAX_BLE_DEVICES = 8;//8
int connectedBLE[MAX_BLE_DEVICES+1] = {0, 0, 0, 0, 0, 0, 0, 0, 999};//MAX_BLE_DEVICES+1
int cnt = 1;
char param[256];
char param1[256];
int result;
char script[200] = "let te;\n\r\n setLCDText('Temp.    ',0,0);\n\r te=readTemp();\n\r
setLCDText_i(te, 12, 0, 3);\n\r if(te<40)\n\r playBuzzer(100,5000,1);";


LOKII lokii;

void setup() {

  param[0] = 0;
  param[1] = 3;
  param[2] = 1;
  param[3] = 1;

  Serial.begin(9600);
  while(!Serial)
  lokii.connect();
}

void loop() {
  lokii.startBLE();
  int bleStatus = lokii.checkBLE();

  if(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE)>= 1 && cnt == 1) //LOKII connected to IOT
devices automatically once the BLE is turned on
  {
    for(int i = 0; i < (MAX_BLE_DEVICES); i++)
    {
      Serial.print("ID");
      Serial.print(i);
      Serial.print(":");
      Serial.println(connectedBLE[i]);
      Serial.println("connected");
      if(connectedBLE[i])
      {
        Serial.println(lokii.setSmartIOTAction(i,script,200));
      }
    }
    Serial.println(lokii.connectIOT(MAX_BLE_DEVICES,connectedBLE));

    cnt++;
  }


}
```

Arduino_VideoStreaming.ino

To do video streaming, LOKII shall set the camera mode to stream mode and connect to WIFI initially (By the default settings of LOKII, the WIFI is enabled) for linking to the LOKII app, which is only available on Android. Two resolutions can be chosen within stream mode, 480*360 and 1280*720. After these setups were completed, users shall connect the device to the same network for connecting the app to the IP address of LOKII. Tap the "CTR switch" to connect the TCP server and then the "SHOW/HIDE AV". Supposedly, a streaming video will be on afterward. Users are able to download this app from this website https://btobsteam.com/resources. This example shows how to do a video streaming for 60 seconds.

```
#include <lokii.h>

#define BLOCKING 1
#define NOBLOCKING 0

LOKII lokii;

// WIFI
uint8_t wifiStatus[2] = {0, 0};

void readWifiStatus(int prtAnyway)
{
   uint8_t router, cloud;

   lokii.getWIFI(&router, &cloud);
   if ( prtAnyway == 1 || wifiStatus[0] != router || wifiStatus[1] != cloud)
   {
      wifiStatus[0] = router;
      wifiStatus[1] = cloud;
      Serial.printf("\nwifi status : %d  %d \n", wifiStatus[0], wifiStatus[1]);
   }
}

void setup(){

  Serial.begin(9600);
  //while(!Serial);

  lokii.connect();
}

void loop(){

  uint8_t router, cloud;

  lokii.playTTS("video streaming Demo" , L_DEFAULT ,5 ,5 ,E_NATURAL);
  lokii.setCameraMode(L_CAM_V_STREAM_1280X720); // 10 seconds

  lokii.setWIFI(1, 1);
  Serial.println("Connecting WIFI and Cloud");
  while ( wifiStatus[0] != 1)
  {
     readWifiStatus(0);
  }
  Serial.printf("\nwifi status : %d  %d \n", wifiStatus[0], wifiStatus[1]);

  delay(60000);

  lokii.setCameraMode(L_CAM_V_STREAM_STOP);
  lokii.playTTS("video streaming stop" , L_DEFAULT ,5 ,5 ,E_NATURAL);

  while (1)
     delay(1000);
}
```

Arduino_CloudpushIOTToWIFI.ino

Users can read connected SMART_IOTs' data in real time via Cloud. LOKII shall connect to WIFI for connecting Cloud and SMART_IOTs via BLE. Please log into your account on https://btobsteam.com/cloud after purchasing the Cloud feature on https://btobsteam.com/lokii-ce. For more information, please contact us (enquiry@centek.com.hk).

```
#include <lokii.h>

#define BLOCKING 1
#define NOBLOCKING 0


LOKII lokii;

// WIFI
uint8_t wifiStatus[2] = {0, 0};
const int MaxdeviceCount = 8;
int deviceList[MaxdeviceCount+1] = {0,0,0,0,0,0,0,0,0};
uint8_t WIFIStatus;
uint8_t cloudStatus;

/*
void readWifiStatus(int prtAnyway)
{
   uint8_t router, cloud;

   lokii.getWIFI(&router, &cloud);
   if ( prtAnyway == 1 || wifiStatus[0] != router || wifiStatus[1] != cloud)
   {
      wifiStatus[0] = router;
      wifiStatus[1] = cloud;
      Serial.printf("\nwifi status : %d  %d \n", wifiStatus[0], wifiStatus[1]);
   }
}
*/

void setup(){

  Serial.begin(9600);
  while(!Serial)
  lokii.connect();

}

void loop(){

  uint8_t router, cloud;

  lokii.startBLE();
  if(lokii.connectIOT(MaxdeviceCount, deviceList) >= 1)
  {
    for(int i = 0; i < MaxdeviceCount; i++)
    {
      if(deviceList[i])
      {
        Serial.println(i);
        Serial.println("connected");
      }
    }
    lokii.getWIFI(&WIFIStatus, &cloudStatus);
    while(cloudStatus == 0 || WIFIStatus == 0)
    {
     while(wifiStatus == 0)
     {
      Serial.println("Connecting to WIFI");
        lokii.setWIFI(1, 0);
     }
```

```
    if(cloudStatus == 0)
    {
     Serial.println("Connecting to Cloud");
       lokii.connectCloud();
    }
    lokii.getWIFI(&WIFIStatus, &cloudStatus);
   }
   Serial.println("WIFI and Cloud connected");
   /*
   while ( wifiStatus[1] != 1)
   {
     readWifiStatus(0);
   }
   Serial.printf("\nwifi status : %d  %d \n", wifiStatus[0], wifiStatus[1]);
   */
   lokii.pushIOTToWIFI(0, 1);
   lokii.playTTS("Data uploaded!",0,5,1,0);

 }
 while (1)
    delay(60000);
    lokii.disconnectCloud();
    lokii.getWIFI(&WIFIStatus, &cloudStatus);
    Serial.println((String)"Cloud Status: " + cloudStatus);
}
```

ExtenderTest.ino

This example shows how to connect LOKII I/O Extender with SMART_Arduino, and use the following functions to read and write GPIO ports status. Those functions are instance methods of class "IOExtender".

```
#include <lokii.h>


LOKII lokii;


#define N55_CS   11
#define N55_MOSI  9
#define N55_MISO 10
#define N55_SCK   6

IOExtender ioExt;

// IOExtender class functions prototype
// ioNum : 0~16 = D0 ~ D16


// software spi pins init
//void initN55SoftSPI(int softCSPin, int softMISOPin, int softMOSIPin, int softCLKPin);

// set IO to INPUT or OUTPUT
// in_or_out : 0 = INPUT PULL HIGH, 1 = OUTPUT LOW
//void setGPIO(uint8_t ioNum, uint8_t in_or_out);

// read single IO status by IO number
//uint8_t readGPIO(uint8_t ioNum);

// low_or_high : 0 = LOW, 1=HIGH
//void writeGPIO(uint8_t ioNum , uint8_t low_or_high);



void writeGPIO(uint8_t ioNum , uint8_t buf)
{
   ioExt.writeGPIO(ioNum, buf);
```

```
}

void setGPIO(uint8_t ioNum, uint8_t in_or_out)
{
   ioExt.setGPIO(ioNum, in_or_out);
}

void syncGPIO(void)
{
   ioExt.syncGPIO();
}

uint8_t readGPIO(uint8_t ioNum)
{
   return ioExt.readGPIO(ioNum);
}

int extenderSPISelfTest(void)
{
   return ioExt.n55SPISelfTest();
}

int lcdPrint(String showString, int lineNum)
{
   return lokii.lcdPrint(showString, lineNum);
}



void setup() {
  // put your setup code here, to run once:

  Serial.begin(115200);
  //while (!Serial);

  delay(100);// first print need delay after Serial.begin
  Serial.println(F("start N55 test"));

  ioExt.initN55SoftSPI(N55_CS, N55_MISO, N55_MOSI, N55_SCK);

  // optional
  if (extenderSPISelfTest() == LOKI_REPLY_OK)
     Serial.println(F("N55 SPI test SUCCESS"));
  else
     Serial.println(F("N55 SPI test FAIL"));


  // N55 work with LOKII class test
  lokii.connect();
}


void loop() {

  int i;
  int changed;
  uint8_t cur[4];
  uint8_t last[4] = {0xFF, 0xFF, 0xFF, 0xFF};

  for (i = 0; i < 4; i++)
  {
     setGPIO(i,0);   // INPUT PULL HIGH
  }

#if 1

  while (true) {

    syncGPIO();

    changed = 0;
```

```
   for (i = 0; i < 4; i++)
   {
      cur[i] = readGPIO(i);
      if ( last[i] == 1 &&
          cur[i] == 0)
      {
         Serial.print(F("D"));
         Serial.print(i);
         Serial.println(F(" = LOW"));

         lcdPrint("LOW ", 5);
         changed = 1;
      }
      if ( last[i] == 0 &&
          cur[i] == 1)
      {
         Serial.print(F("D"));
         Serial.print(i);
         Serial.println(F(" = HIGH"));
         lcdPrint("HIGH", 5);
         changed = 1;
      }
      last[i] = cur[i];
   }
   if (changed)
      delay(10); // debounce, no debounce will make the print buffer overflow
  }

#endif

}
```

QRdetect.ino

LOKII is looking for a QR code image (QR Code Version 2 format), print out the QR code's message on LCD and speak out using Text-To-Speech function.

```
#include <lokii.h>

#define BLOCKING 1
#define NOBLOCKING 0

//void getQRResult(char *qrString );
//bool waitForQRResult(bool isBlocking);


unsigned long StartTime = 0;

char qrCode[128];

LOKII lokii;

void QRDetectSample() {


  if (lokii.waitForQRResult() == true) {

    qrCode[0] = 0;
    lokii.getQRResult(qrCode);
    lokii.setCameraMode(L_CAM_PREVIEW);
    lokii.playTTS(qrCode, L_DEFAULT ,5 ,5 ,E_NATURAL);
    Serial.println(qrCode);
    lokii.setCameraMode(L CAM QRCODE);


  }
}
```

```
void setup(){

  Serial.begin(115200);

  lokii.connect();

}

void loop(){

  lokii.playTTS("QR Code Demo" , L_DEFAULT ,5 ,5 ,E_NATURAL);
  delay(3000);
  lokii.setCameraMode(L_CAM_QRCODE);
  while (true)
  {
     QRDetectSample();
  }


}
```