Bluetooth Low Energy: CC2540 Technical Training

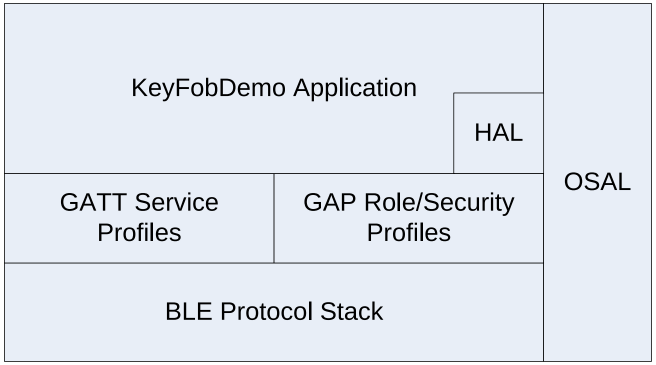
Software Overview

Agenda

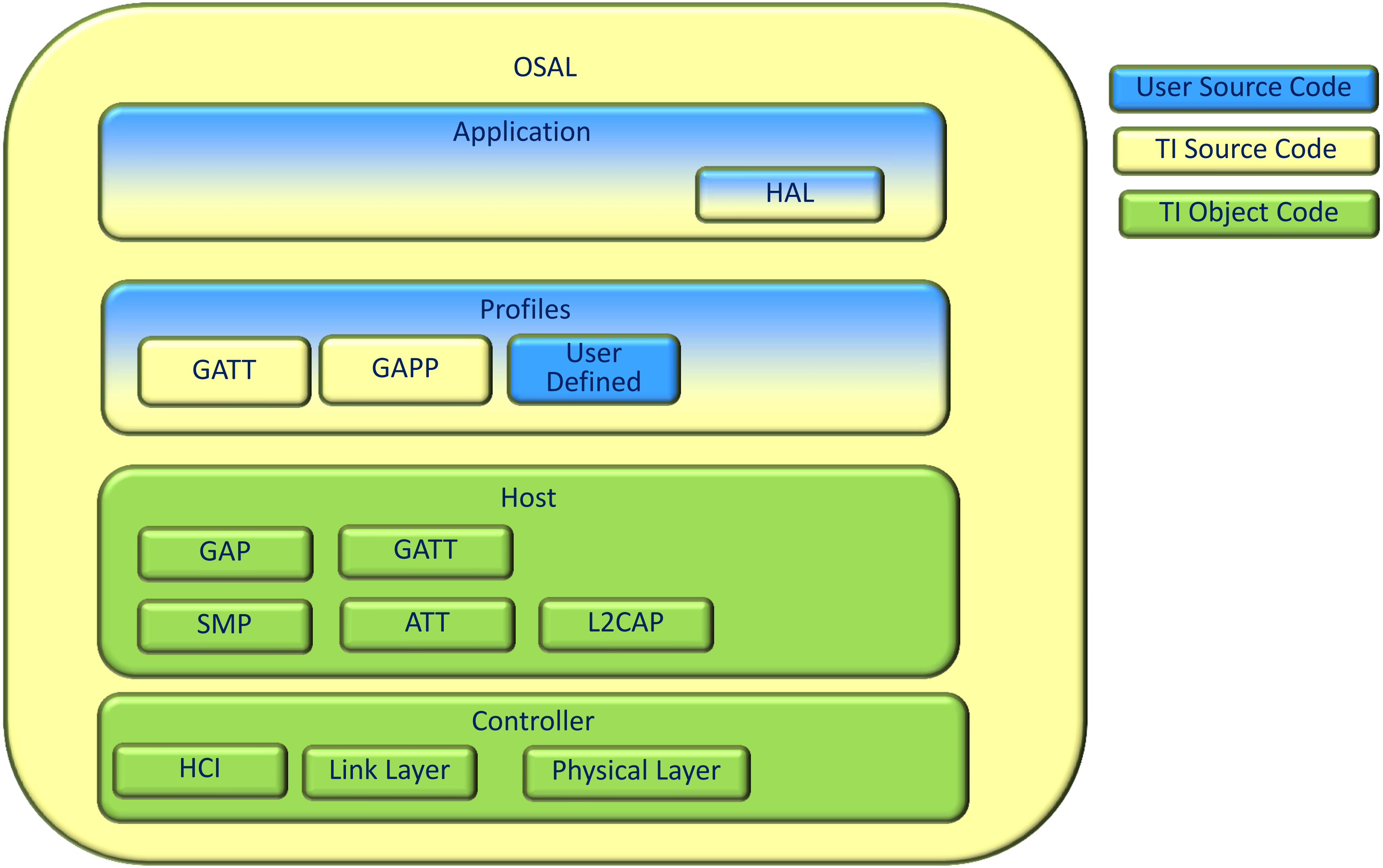
- CC2540 Software Overview
  - CC2540 Software - Architecture and Structure
  - OSAL (Operating System Abstraction Layer) - Task setup and initialization, events and processing, messaging and memory managers
  - HAL (Hardware Abstraction Layer)
  - GAP Role Profiles - Peripheral and Peripheral / Broadcaster role profiles
  - GATT Profiles - Structure and format, initialization, application callbacks

Software Overview

- Five major parts of the software:
  - Operating System Abstraction Layer (OSAL)
  - Hardware Abstraction Layer (HAL)
  - Application
  - BLE Protocol Stack
  - Profiles: GAP Role, GAP Security, and GATT Services



Software Overview



TI confidential information - Strictly Private

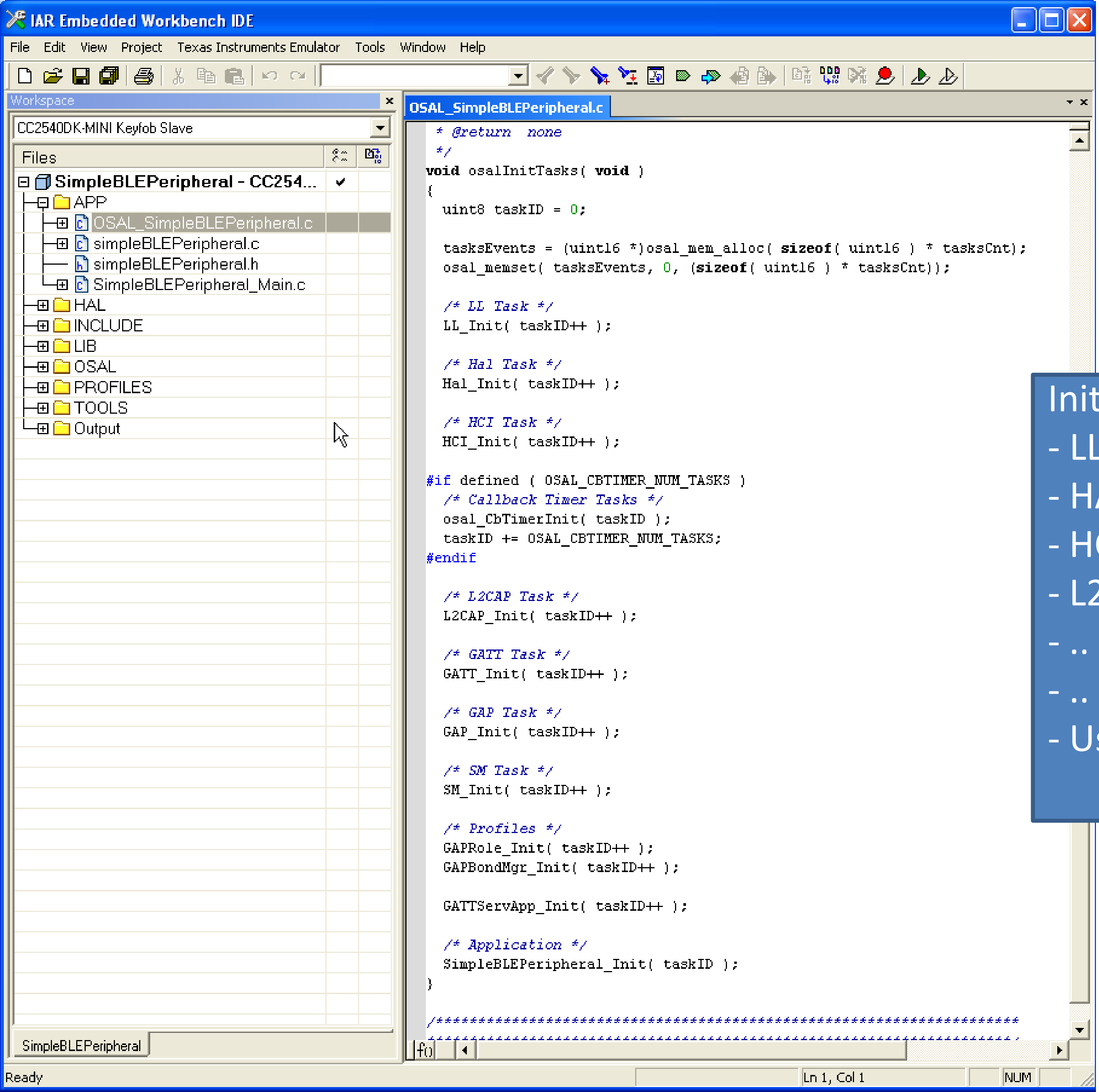
Operating System Abstraction Layer (OSAL)

- The software architecture of the CC2540 is based around the Operating System Abstraction Layer (OSAL)
- The OSAL is not an actual operating system (OS) in the traditional sense, but rather a control loop that allows software to setup execution of events
- Each subsystem of the software runs as an OSAL task, and has a unique task identifier (ID)
- The lower the task ID, the higher the priority for the task
- The KeyFobDemo Project has 12 OSAL tasks (task ID in parenthesis):
  - Link Layer (0)
  - HAL (1)
  - HCI (2)
  - OSAL Callback Timer (3)
  - L2CAP (4)
  - GATT (5)
  - GAP (6)
  - SM (7)
  - Peripheral Role Profile (8)
  - GAP Bond Manager (9)
  - GATT Server (10)
  - SimpleKeys Application (11)

OSAL: Task Setup

- Each task is required to have two functions:
  - Initialization (example: SimpleKeys_Init)
  - Event Handler (example: SimpleKeys_ProcessEvent)
- Every application that uses the OSAL must define a function called "osalInitTasks" (void parameters and void return)
- This function calls each task's intialization function, and sets up it's task ID
- Every application must also create a global variable called "tasksArr", which is array consisting of one pointer to each task's event handler function
- The order of the elements in the array must be exactly the same as the order of the task IDs
- Application must also create a global variable called "tasksEvents", which is an array consisting of one uint16 value for each task
  - All elements of the tasksEvents array must be initialized to zero
  - Each element of the array represents the pending events for that task

OSAL


IAR Embedded Workbench IDE
File
Edit
View
Project
Texas Instruments Emulator
Tools
Window
Help
Workspace
CC2540DK-MINI Keyfob Slave
Files
SimpleBLEPeripheral - CC254...
APP
OSAL_SimpleBLEPeripheral.c
simpleBLEPeripheral.c
simpleBLEPeripheral.h
SimpleBLEPeripheral_Main.c
HAL
INCLUDE
LIB
OSAL
PROFILES
TOOLS
Output
SimpleBLEPeripheral
OSAL_SimpleBLEPeripheral.c
Ready
Ln 1, Col 1
NUM

```
 * @return  none
 */
void osalInitTasks( void )
{
  uint8 taskID = 0;

  tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);
  osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));

  /* LL Task */
  LL_Init( taskID++ );

  /* Hal Task */
  Hal_Init( taskID++ );

  /* HCI Task */
  HCI_Init( taskID++ );

#if defined ( OSAL_CBTIMER_NUM_TASKS )
  /* Callback Timer Tasks */
  osal_CbTimerInit( taskID );
  taskID += OSAL_CBTIMER_NUM_TASKS;
#endif

  /* L2CAP Task */
  L2CAP_Init( taskID++ );

  /* GATT Task */
  GATT_Init( taskID++ );

  /* GAP Task */
  GAP_Init( taskID++ );

  /* SM Task */
  SM_Init( taskID++ );

  /* Profiles */
  GAPRole_Init( taskID++ );
  GAPBondMgr_Init( taskID++ );

  GATTServApp_Init( taskID++ );

  /* Application */
  SimpleBLEPeripheral_Init( taskID );
}

/*********************************************************************
```

InitTask
- LL_Init
- HAL
- HCI
- L2CAP
- ..
- ..
- User Application

OSAL : Events

- An OSAL “event” is a scheduled process for a task to run
- Any OSAL task can define up to 15 events in addition to the mandatory SYS_EVENT_MSG event (0x8000), which is used for OSAL messaging
- Events can be set using one of two OSAL API functions:
  - osal_set_event – immediately schedules the event to occur
  - osal_start_timerEx – schedules the event to occur at a specific time in the future (set in milliseconds)
- An event set up using osal_start_timerEx can be cancelled by calling OSAL API function osal_stop_timerEx
- Each element in the tasksEvents array acts as a 16-bit mask for each task, with any set bit indicating that a specific event is scheduled for that task
- In example below, bit 8 of task 1 is set, indicating that the event with a defined mask value of 0x0100 should be processed

	MSB															LSB
Task 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Task 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

OSAL: Main Loop

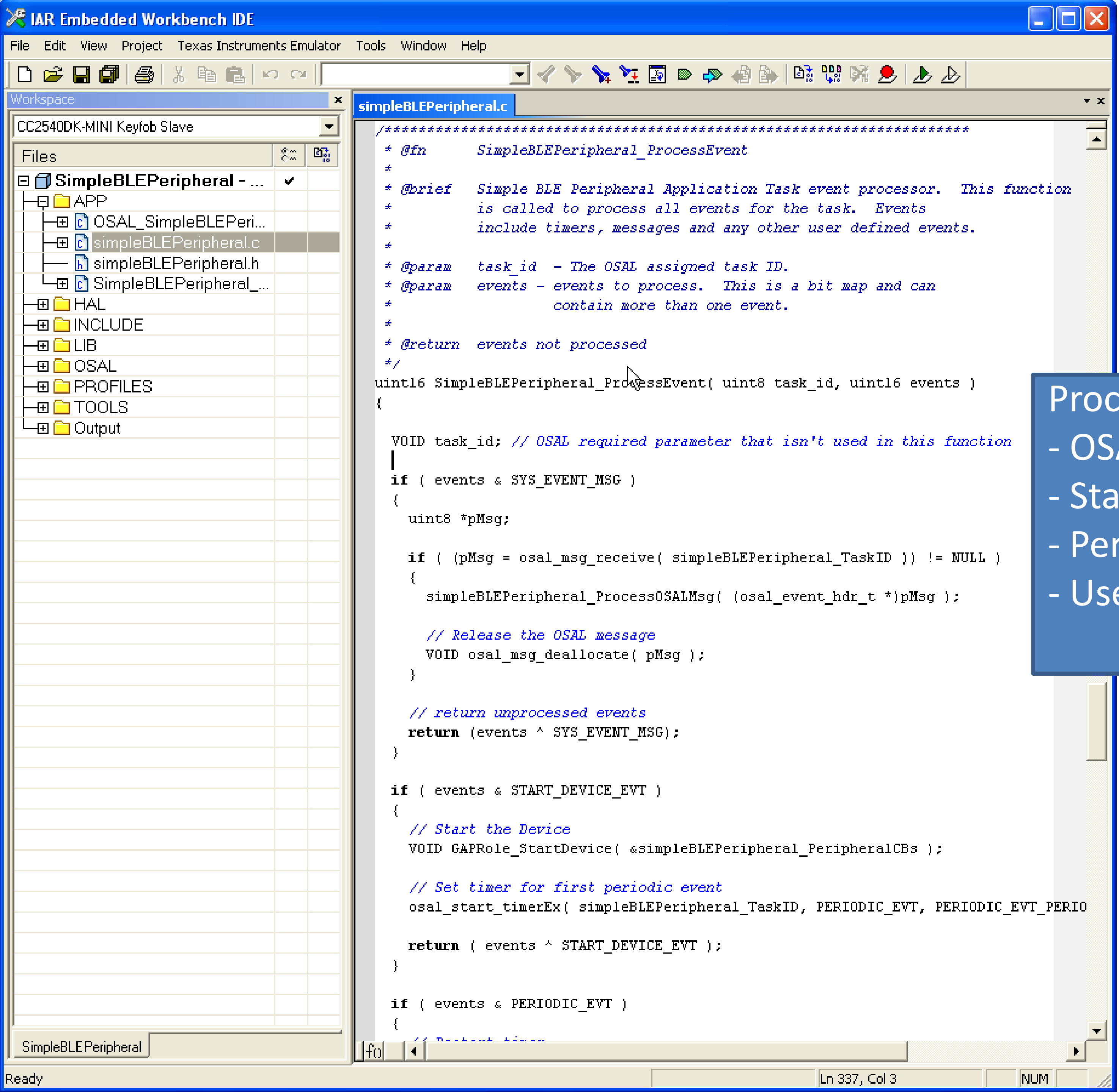
OSAL

- The OSAL main loop is run when the function osal_start_system is called
- The loop checks each element of the tasksEvents array for a non-zero value (which would indicate that at least one event bit is set)
- The loop always processes a pending event with a lower task ID first
- When a non-zero value is found, OSAL will call the task's event handler function, using the pointer from tasksArr
- After the event is processed, it is up to the task to clear the event bit; if it doesn't get cleared the OSAL will keep calling the event handler function
- If every single element in the tasksEvents array has a zero value (meaning that none of the tasks have any events scheduled) the OSAL puts the processor into power savings mode, in which memory remains stored and timers continue running
- Processor will wake up when an interrupt occurs or when an OSAL timer schedules a task event

OSAL: Message Management

OSAL

- OSAL provides a system for different subsystems of the software to communicate with each other by sending or receive messages
- Messages can contain any type of data and can be any size
- Process to send a message:
  - Allocate memory using osal_msg_allocate
  - Copy data into allocated memory space, including a header indicating the type
  - Call osal_msg_send, indicating destination task for the message
- OSAL signals to receiving task that a message is arriving by setting the SYS_EVENT_MSG flag for that task
- The receving task's event handler function retrieves the data and calls it's local message processing function (example: keyfobapp_ProcessOSALMsg)
- The receiving task must deallocate the memory using the function osal_msg_deallocate



```
/*********************************************************************
 * @fn      SimpleBLEPeripheral_ProcessEvent
 *
 * @brief   Simple BLE Peripheral Application Task event processor.  This function
 *          is called to process all events for the task.  Events
 *          include timers, messages and any other user defined events.
 *
 * @param   task_id  - The OSAL assigned task ID.
 * @param   events - events to process.  This is a bit map and can
 *                   contain more than one event.
 *
 * @return  events not processed
 */
uint16 SimpleBLEPeripheral_ProcessEvent( uint8 task_id, uint16 events )
{

  VOID task_id; // OSAL required parameter that isn't used in this function

  if ( events & SYS_EVENT_MSG )
  {
    uint8 *pMsg;

    if ( (pMsg = osal_msg_receive( simpleBLEPeripheral_TaskID )) != NULL )
    {
      simpleBLEPeripheral_ProcessOSALMsg( (osal_event_hdr_t *)pMsg );

      // Release the OSAL message
      VOID osal_msg_deallocate( pMsg );
    }

    // return unprocessed events
    return (events ^ SYS_EVENT_MSG);
  }

  if ( events & START_DEVICE_EVT )
  {
    // Start the Device
    VOID GAPRole_StartDevice( &simpleBLEPeripheral_PeripheralCBs );

    // Set timer for first periodic event
    osal_start_timerEx( simpleBLEPeripheral_TaskID, PERIODIC_EVT, PERIODIC_EVT_PERIO

    return ( events ^ START_DEVICE_EVT );
  }

  if ( events & PERIODIC_EVT )
  {
    // Restart timer
```

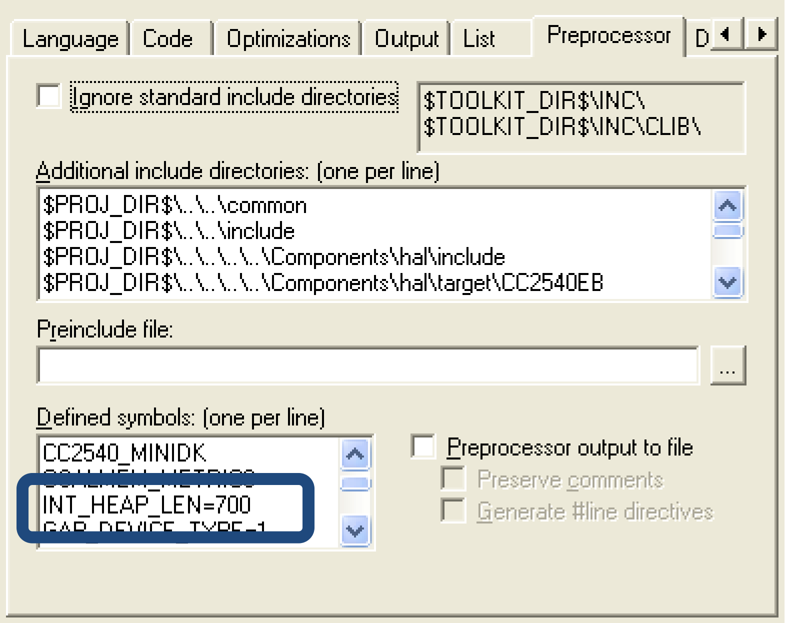
OSAL

ProcessEvent
- OSAL Msg
- StartDevice
- Periodic Event
- UserDefined

OSAL: Memory Management

OSAL

- OSAL APIs for memory allocation and deallocation:
  - osal_mem_alloc
  - osal_mem_free
- Heap size set with preprocessor defined symbol INT_HEAP_LEN
- If heap size is set too high, CC2540 may run out of memory
- Check map file to verify that memory has not exceeded limits (8kB)



□ Ignore standard include directories

```
$TOOLKIT_DIR$\INC\
$TOOLKIT_DIR$\INC\CLIB\
```

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\common
$PROJ_DIR$\..\..\include
$PROJ_DIR$\..\..\..\..\Components\hal\include
$PROJ_DIR$\..\..\..\..\Components\hal\target\CC2540EB
```

Preinclude file:

Defined symbols: (one per line)

```
CC2540_MINIDK
OSALMEM_METRICS
INT_HEAP_LEN=700
GAP_DEVICE_TYPE=1
```

□ Preprocessor output to file

□ Preserve comments

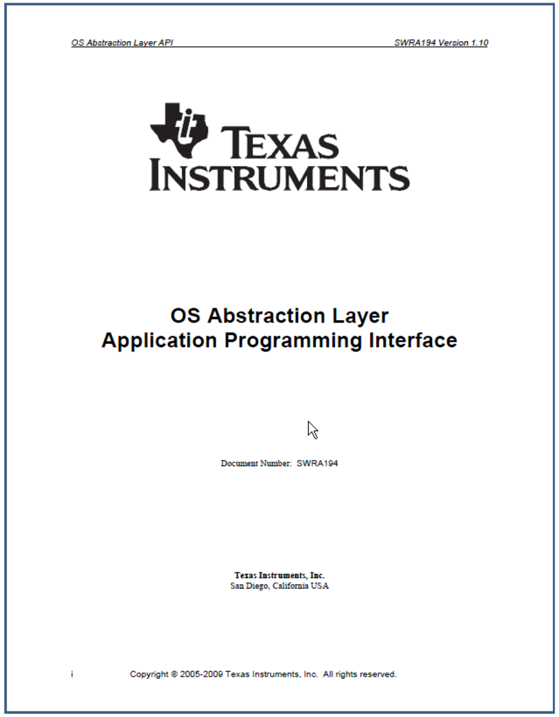
□ Generate #line directives

OSAL: Files and Key API's

OSAL

- Key Files:
  - osal.c – API's for OSAL
  - osal.h – OSAL API declarations
- Key API's:
  - osal_init_system – initializes OSAL; must be called in main
  - osal_start_system – starts the OSAL main loop
  - osal_set_event – sets an OSAL event for a task
  - osal_start_timerEx – sets an OSAL event for a task at a scheduled moment in time
  - osal_stop_timerEx – cancels an existing OSAL event that was scheduled using osal_start_timerEx
  - osal_msg_allocate – dynamically allocates memory for an OSAL message
  - osal_msg_send – sends an OSAL message to a specific task
  - osal_msg_deallocate – deallocates an OSAL message (call this from receiving task)
  - osal_mem_alloc – dynamically allocates memory
  - osal_mem_free – free previously allocated memory
- The following OSAL function must be defined by the application:
  - OsalInitTasks – set up task ID's for each task used by OSAL
- Additional information on the OSAL can be found in the OSAL API guide:

C:\Texas Instruments\BLE-CC2540\Documents\osal\OSAL API.pdf



OS Abstraction Layer API SWRA194 Version 1.10

TEXAS INSTRUMENTS

OS Abstraction Layer
Application Programming Interface

Document Number: SWRA194

Texas Instruments, Inc.
San Diego, California USA

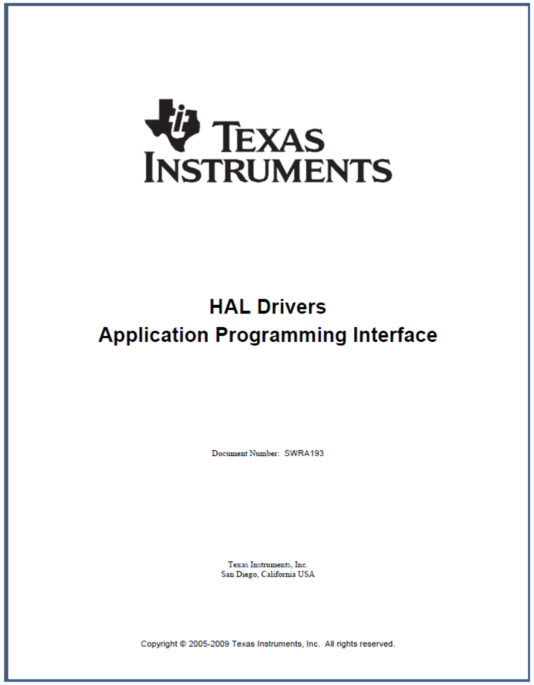
i Copyright © 2005-2009 Texas Instruments, Inc. All rights reserved.

Hardware Abstraction Layer (HAL)

HAL

- The Hardware Abstraction Layer (HAL) provides an application programming interface to hardware-related functions
  - ADC
  - UART
  - SPI
  - Flash
  - Timers
  - Keys
  - LCD Driver

C:\Texas Instruments\BLE-CC2540\Documents\hal\HAL Driver API.pdf



Texas Instruments

HAL Drivers
Application Programming Interface

Document Number: SWRA193

Texas Instruments, Inc.
San Diego, California USA

Copyright © 2005-2009 Texas Instruments, Inc. All rights reserved.

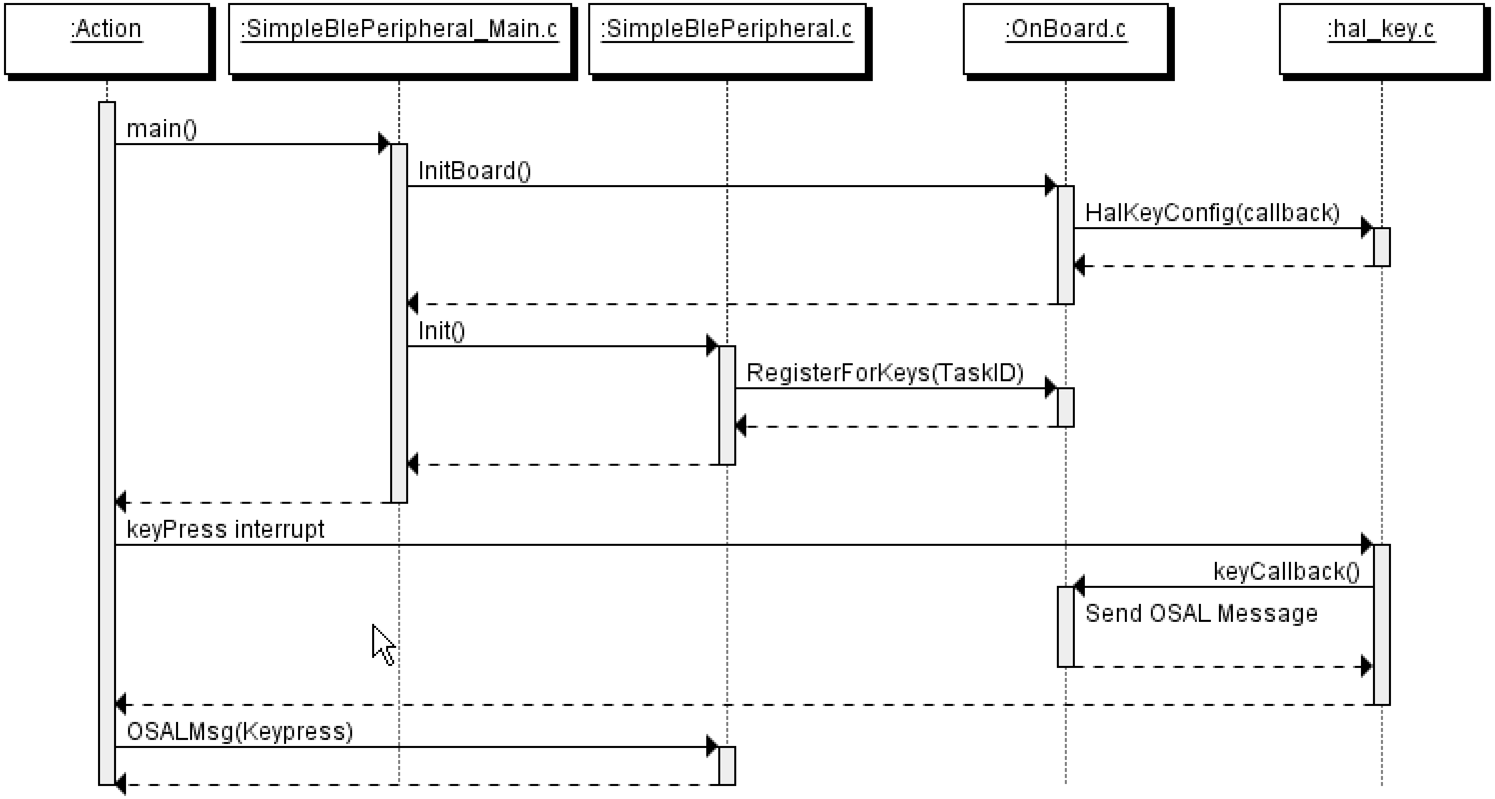
HAL

HAL: Key Handling

- Application registers with HAL during intialization by calling function RegisterForKeys, allowing HAL to know the application task ID
- Key presses are handled by HAL using interrupts
- When the state of one of the keys changes, an OSAL message with type KEY_CHANGE is sent to the application
- Application calls local function keyfobapp_HandleKeys
  - Checks which keys were pressed
  - If device is not connected, checks peripheral role profile to see whether device is advertising or not, and toggles advertisements on or off
  - Sets the state of the keys value in the Simple Keys profile using the function SimpleKeys_SetParameter
- If the device is in a connected state and notifications of the key presses have been enabled, the keyfob will send a GATT notification to the master device over the air (more information on this later)

HAL: Keypress example

HAL



HAL: Other Usage Examples

HAL

KeyFobDemo Application

- Key press notifications
- Buzzer (GPIO)
- Battery percentage measurement (ADC)
- Accelerometer data notification (SPI)

Source Code for KeyFob Demo available on TI Wiki

http://processors.wiki.ti.com/index.php/Category:KeyFobDemo

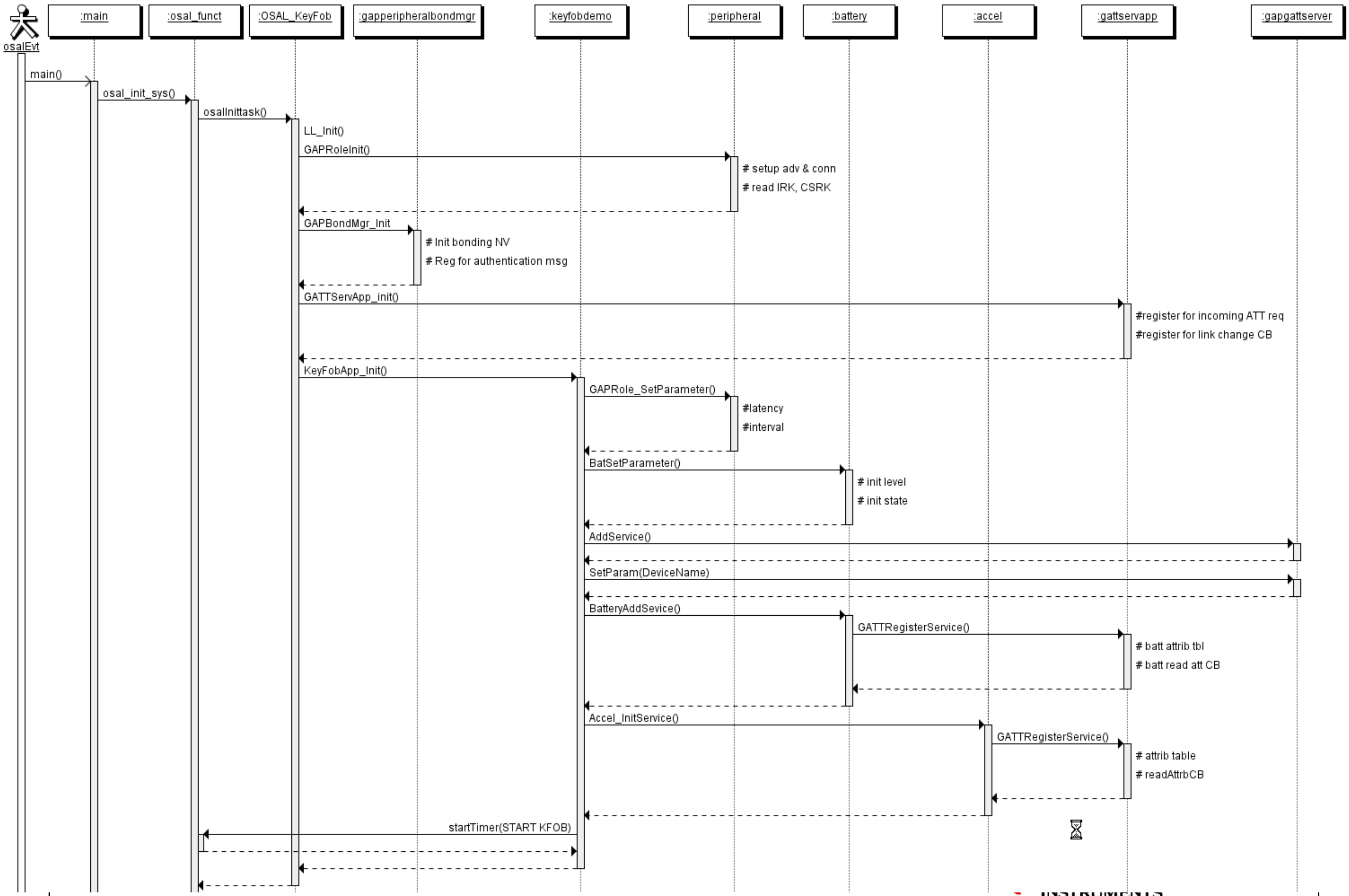
SimplePeripheral:Startup

Application

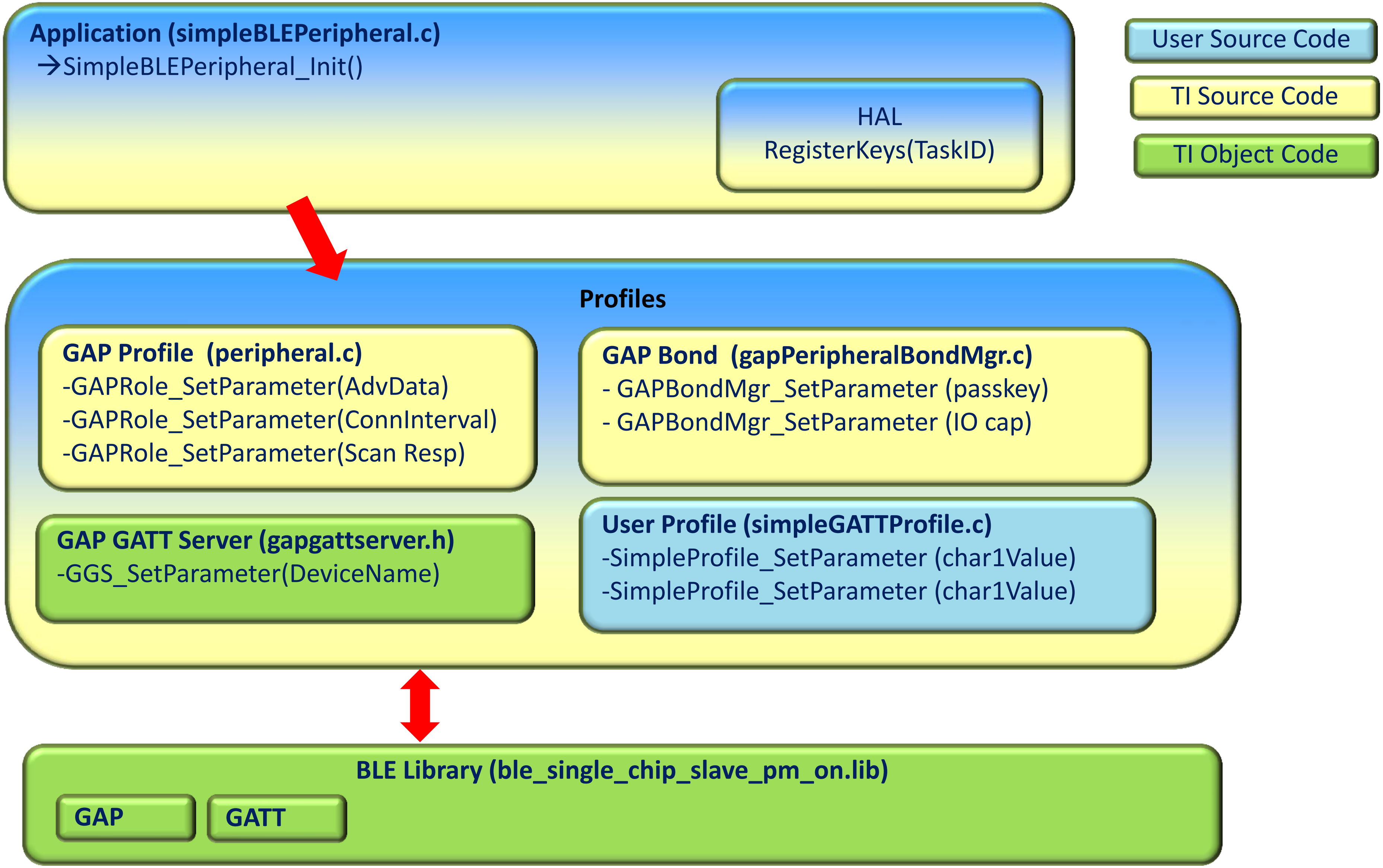
- The application starts with the main function in the file KeyFob_Main.c
- The BleSimplePeripheral_App_Init function is called during task initialization
  - Sets Peripheral Role profile initial parameters
  - Sets GATT profile initial parameters
  - Initializes each GATT service
  - Registers with HAL to receive OSAL message when key presses occur
  - Uses osal_start_timerEx to set a KEYFOB_START_DEVICE_EVT after a 500ms delay

- After the 500ms delay, application task event process handler function gets called due to KEYFOB_START_DEVICE_EVT flag getting set
  - Application callbacks registered with proximity and accelerometer profiles
  - KEYFOB_START_DEVICE Event flag cleared

KeyFob:Startup



Application Startup (set values)



Application (simpleBLEPeripheral.c)
→SimpleBLEPeripheral_Init()

HAL
RegisterKeys(TaskID)

User Source Code

TI Source Code

TI Object Code

Profiles

GAP Profile (peripheral.c)
-GAPRole_SetParameter(AdvData)
-GAPRole_SetParameter(ConnInterval)
-GAPRole_SetParameter(Scan Resp)

GAP Bond (gapPeripheralBondMgr.c)
- GAPBondMgr_SetParameter (passkey)
- GAPBondMgr_SetParameter (IO cap)

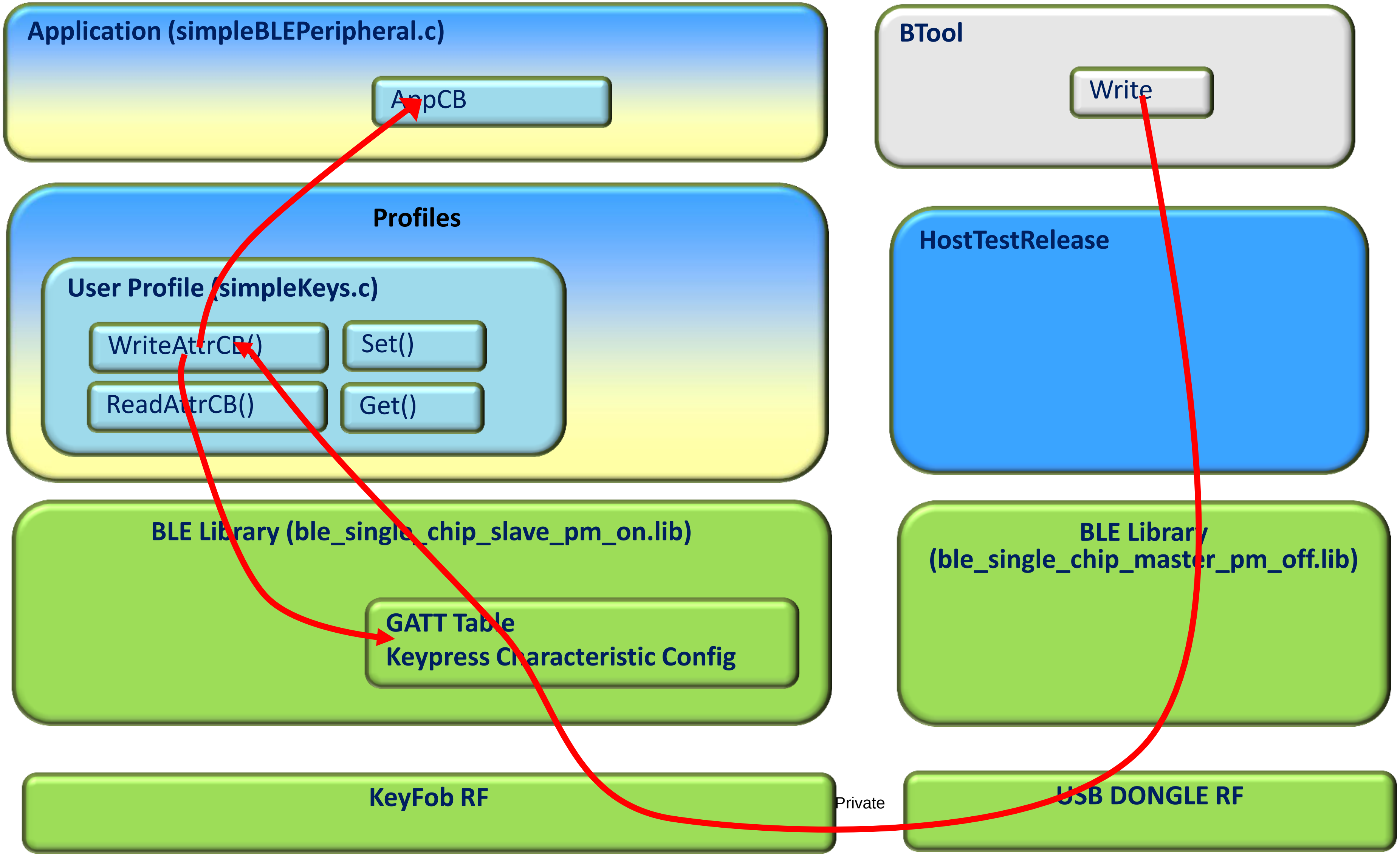
GAP GATT Server (gapgattserver.h)
-GGS_SetParameter(DeviceName)

User Profile (simpleGATTProfile.c)
-SimpleProfile_SetParameter (char1Value)
-SimpleProfile_SetParameter (char1Value)

BLE Library (ble_single_chip_slave_pm_on.lib)

GAP **GATT**

Application – Turn on Notifications



Application (simpleBLEPeripheral.c)

AppCB

BTool

Write

Profiles

User Profile (simpleKeys.c)

WriteAttrCB()

Set()

ReadAttrCB()

Get()

HostTestRelease

BLE Library (ble_single_chip_slave_pm_on.lib)

GATT Table
Keypress Characteristic Config

BLE Library
(ble_single_chip_master_pm_off.lib)

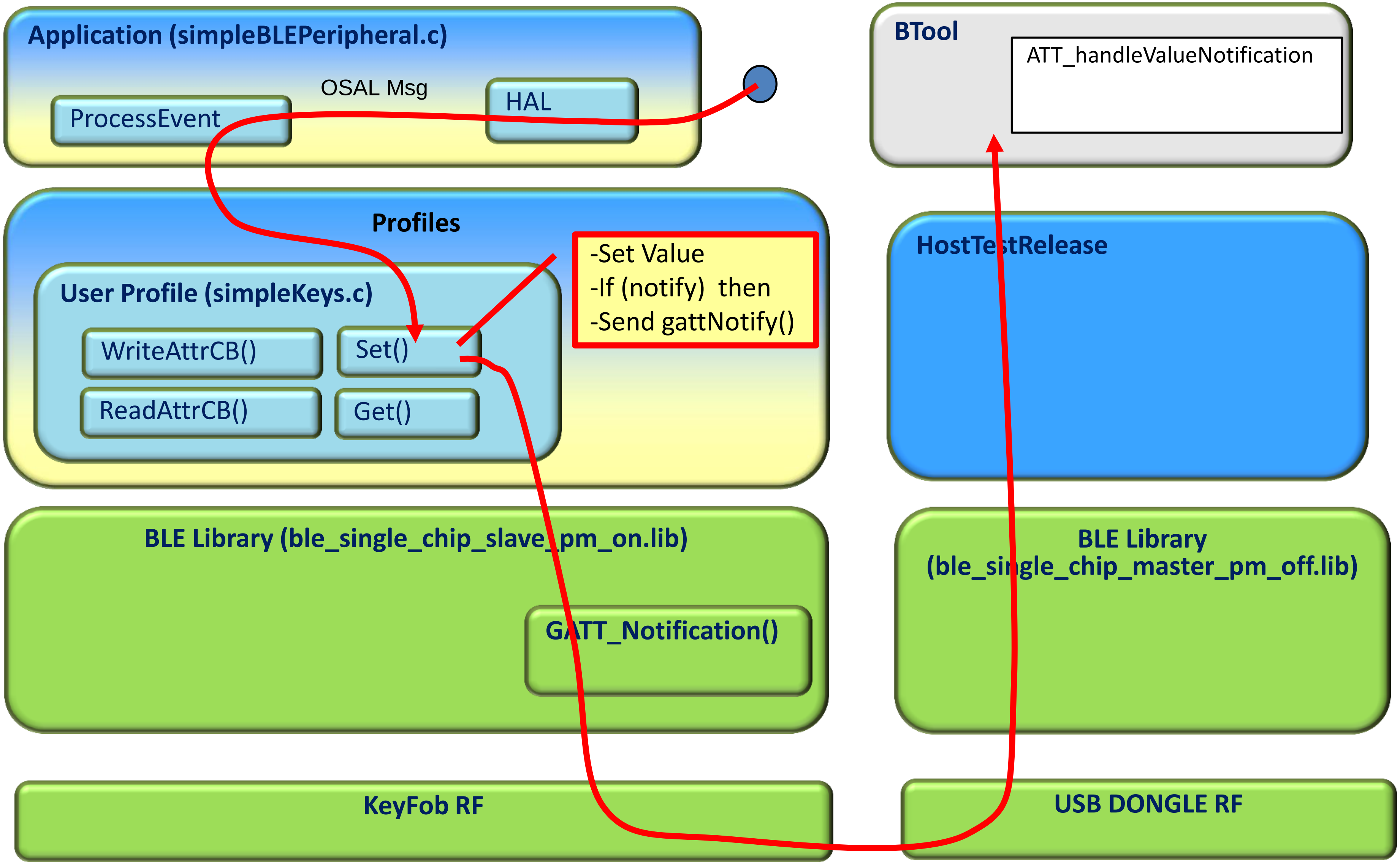
KeyFob RF

USB DONGLE RF

Private

Texas Instruments

Application – Keypress Notification

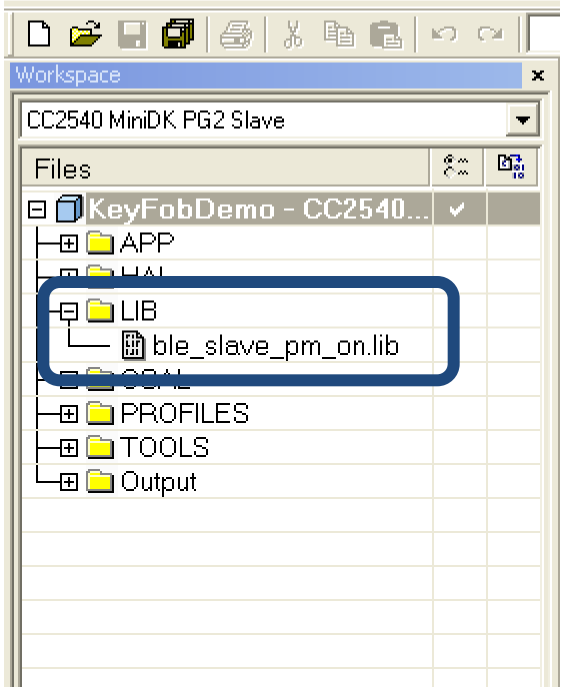


Application (simpleBLEPeripheral.c)
OSAL Msg
ProcessEvent
HAL
BTool
ATT_handleValueNotification
Profiles
User Profile (simpleKeys.c)
WriteAttrCB()
Set()
ReadAttrCB()
Get()
HostTestRelease
BLE Library (ble_single_chip_slave_pm_on.lib)
GATT_Notification()
BLE Library
(ble_single_chip_master_pm_off.lib)
KeyFob RF
USB DONGLE RF

-Set Value
-If (notify) then
-Send gattNotify()

BLE Stack

- The Beta BLE protocol stack is based on the approved Bluetooth Core specification version 4.0 (June 30, 2010)
- Protocol stack provided as a single library file in KeyFobDemo application (three versions provided: one for each hardware platform)
- Application usually does not need to directly call protocol stack API's
- Profiles provide a means for application to send and receive control messages and data with stack



TI confidential information - Strictly Private

Profiles Overview

- Profiles provide a layer of software between the application and the BLE protocol stack
- Allow developer to perform basic BLE functions without having in-depth knowledge of the stack
- Directly communicate with the top two layers of the BLE stack
  - **GAP Peripheral Role Profile** – Handles advertisements, scan requests, connections, and connection parameters
  - **GAP Peripheral Bond Manager** – Handles responses to pairing and bonding requests, and the storage and management of security keys
  - **GATT Profiles** – Maintain GATT attributes in table, processing of read and write requests, and notifications

GAP Peripheral Role Profile: Purpose

- Allows device to act as a GAP peripheral and perform the following:
  - Turn advertising on and off
  - Send connectable advertisements and accept connection requests
  - Request automatic updates of link-layer connection parameters to a central device:
    - Connection interval
    - Slave latency
    - Supervision timeout
  - Notify application of connection state changes

GAP Peripheral Role Profile: Public Functions

- Peripheral Role Profile is an OSAL task, and contains initialization and event processing functions called by OSAL:
  - GAPRole_Init
  - GAPRole_ProcessEvent
- Profile contains several parameters, accessed through:
  - GAPRole_SetParameter
  - GAPRole_GetParameter
- Initialization from application:
  - GAPRole_StartDevice
- Terminate a connection:
  - GAPRole_TerminateConnection

GAP Peripheral Role Profile: Initialization

- OSAL initializes Peripheral Role with call to GAPRole_Init
- Application registers two callback functions with Peripheral Role Profile by passing function pointers as parameter to GAPRole_StartDevice function:
  - peripheralStateNotificationCB – notifies application that the peripheral device has changed GAP states (for example, devices goes from advertising to being in a connection)
  - rssiAvailableCB – notifies application of the RSSI when it becomes available (set to NULL in KeyFobDemo application since it does not use RSSI information)
- When GAPRole_StartDevice is called:
  - Profile signals GAP to begin advertising (if enabled)
  - Profile registers itself with GAP as the task to receive GAP event messages (this allows profile to always know the connection status)
- In KeyFobDemo application, GAPRole_StartDevice is not called until 500ms delay (triggered by KEYFOB_START_DEVICE_EVT)

GAP Peripheral Role Profile: Key Parameters

- GAPROLE_ADVERT_DATA – Advertisement data string
- GAPROLE_SCAN_RSP_DATA – Scan response data string
- GAPROLE_ADVERT_ENABLED – a TRUE or FALSE value indicating if advertising is enabled
- GAPROLE_RSSI_READ_RATE – amount of time (in ms) of RSSI readings
- GAPROLE_PARAM_UPDATE_ENABLE – enabled automatic connection parameter update requests if master establishes a connection with unwanted parameters (TRUE or FALSE)
- GAPROLE_MIN_CONN_INTERVAL – the minimum connection interval for the device (in units of 1.25ms as per link layer specification)
- GAPROLE_MAX_CONN_INTERVAL – the maximum connection interval for the device (in units of 1.25ms as per link layer specification)
- GAPROLE_SLAVE_LATENCY – the connection slave latency setting
- GAPROLE_TIMEOUT_MULTIPLIER – the connection supervision timeout setting

GAP Peripheral Role Profile: Advertisement and Scan Response Data

- The GAPROLE_ADVERT_DATA and GAPROLE_SCAN_RSP_DATA parameters allow application to set the GAP data sent to a central or observer device while in the advertising state

- Data must conform to GAP specification for “AD types”:
  - The first byte contains the length of the data
  - The second byte contains a value indicating the AD type accoring to spec (ex. 0x09 = Local Name, 0x01 = Flags)

GAP Peripheral Role Profile: AD Types Used

- In KeyFobDemo application:
  - Advertisement Data String:

0x0A (length 10)	0x09 (name)	0x50 ‘P’	0x72 ‘r’	0x6F ‘o’	0x78 ‘x’	0x69 ‘i’	0x6D ‘m’	0x69 ‘i’	0x74 ‘t’	0x79 ‘y’

  - Scan Response Data String:

0x02 (2)	0x01 (flags)	0x02 (General Discoverable)

    - By setting “General Discoverable”, device will continuously advertise as long as advertisements are enabled
    - If set to “Limited Discoverable” (0x01), when advertisements are enabled the device will advertise for a limited time, stop for 10 seconds, and repeat

GAP Peripheral Role Profile: After Link Establishment

- Once a connection is established, GAP sends an OSAL message of type GAP_EST_LINK_REQ_EVENT to GAP application (peripheral role profile)
- RSSI read timer starts
- Profile calls the callback function peripheralStateNotificationCB to notify application that GAP state has changed
- Profile checks the connection interval and slave latency setting for the connection, and (if enabled) will send an automatic update request if:
  - Interval falls outside the range set by GAPROLE_MIN_CONN_INTERVAL and GAPROLE_MAX_CONN_INTERVAL parameters
  - OR latency setting does not equal GAPROLE_SLAVE_LATENCY parameter value
  - OR supervision timeout settings does not equal GAPROLE_TIMEOUT_MULTIPLIER parameter value
- Update parameter request sent with connection parameter values in profile
- Profile uses osal_start_timerEx to set the UPDATE_PARAMS_TIMEOUT_EVT OSAL event for itself after a set time (calculated based on the max amount of time)
  - If update parameter response is received before timeout, osal_stop_timerEx called to cancel event
  - If timeout expires before response is received, peripheral device terminates connection

GAP Peripheral Role Profile: RSSI Measurement

- The peripheral role profile can provide RSSI measurements to the application with the callback function rssiAvailableCB (this feature is not used by the KeyFobDemo application)
- RSSI can only be read when device is in a connection
- RSSI value only updated when data is received (in future release, RSSI will update with each link layer connection event)
- GAPROLE_RSSI_READ_RATE parameter sets the amount of time in milliseconds between RSSI reads
- When device enters connected state, profile calls osal_start_timerEx to schedule an RSSI_READ_EVT
- Every time RSSI_READ_EVT occurs:
  - Profile calls HCI_ReadRssiCmd function
  - Peripheral role profile receives OSAL message from GAP (message type HCI_GAP_EVENT_EVENT) containing RSSI reading
  - Profile calls callback function rssiAvailableCB to notify application of value

GAP Peripheral Role Profile: After Link Termination

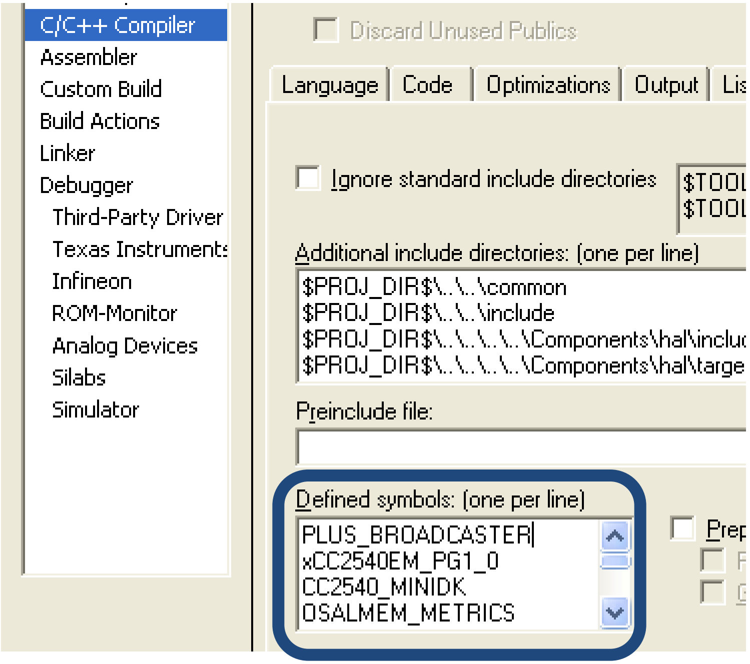
- When a connection is terminated for any reason, GAP sends OSAL message to GAP application (peripheral role profile) of type GAP_TERMINATE_LINK_EVENT

- Profile calls the callback function peripheralStateNotificationCB to notify application that GAP state has changed, and whether the link terminated due to supervision timeout, or due to a terminate link request

- Profile schedules a START_ADVERTISING_EVT using the osal_start_timerEx function, with the amount of time determined by the value of the parameter GAPROLE_ADVERT_OFF_TIME

GAP Peripheral Role Profile: Switching to multi-role profile

- In addition to peripheral role profile, includes a peripheral / broadcaster multi-role profile
- To use multi-role profile:
  - Exclude the files "peripheral.c" and "peripheral.h" from the KeyFobDemo project (right-click on files and select "options" in IAR, then check the box for "Exclude from build")
  - Add the files "peripheralBroadcaster.c" and "peripheralBroadcaster.h" to the project under the "Profiles" group
- In IAR Project options (compiler settings), add the preprocessor defined symbol "PLUS_BROADCASTER"
- All functions have the same names and work identical to the functions in peripheral.c
- Advertisements can now be enabled or disabled by setting GAPROLE_ADVERT_ENABLED parameter value to TRUE while in a connected state
- Advertisements will be non-connectable



Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\common
$PROJ_DIR$\..\..\include
$PROJ_DIR$\..\..\..\..\Components\hal\incluc
$PROJ_DIR$\..\..\..\..\Components\hal\targe
```

Preinclude file:

Defined symbols: (one per line)

```
PLUS_BROADCASTER
xCC2540EM_PG1_0
CC2540_MINIDK
OSALMEM_METRICS
```

GATT Service Profiles: Overview

- Allows device to implement a GATT service:
  - As defined by Bluetooth SIG
  - Custom
- Provides means for application to read and write service data on the attribute table
- Lets a remote GATT client access characteristics through:
  - GATT reads
  - GATT writes
  - GATT notifications and indications
- Verifies the validity of data being written from a remote device
- GATT service profiles typically do not need to be OSAL tasks, and are accessed directly by the protocol stack and by the application
- Most GATT service profiles have a very similar structure
- New GATT service profiles can be easily created by copying an existing profile and renaming variables and functions

GATT Service Profiles: Typical Functions

- Public functions:
  - ProfileName_AddService – registers attribute list and callback functions with GATT server
  - ProfileName_RegisterAppCBs – allows function to register application callback functions with profiles.
  - ProfileName_SetParameter – allows application to set attribute data values; also sends out notifications of characteristics when enabled
  - ProfileName_GetParameter – allows application to get attribute values
- Private GATT server callback functions:
  - profileName_ReadAttrCB – called when a GATT read request is received from a GATT client; returns attribute data to GATT server for read response
  - profileName_ValidateWriteAttrCB – called when a GATT write request is received from a GATT client; validates data being written and writes new value if data is valid; sends write response with appropriate error message if data is invalid

GATT Service Profiles: Structure

- Attribute value variables are defined as static and are local to the module
- Standard UUID’s (from BT SIG) are defined in gatt_uuid.h
- Custom UUID’s are defined in profiles own header file
- In addition to attribute values, profile defines an array of type gattAttribute_t, in which each element contains data related to each attibute:
  - Attribute type (UUID length in bytes and UUID itself)
  - Permissions
  - Handle – profile initializes this to zero, and server updates when building the table
  - Pointer to data value

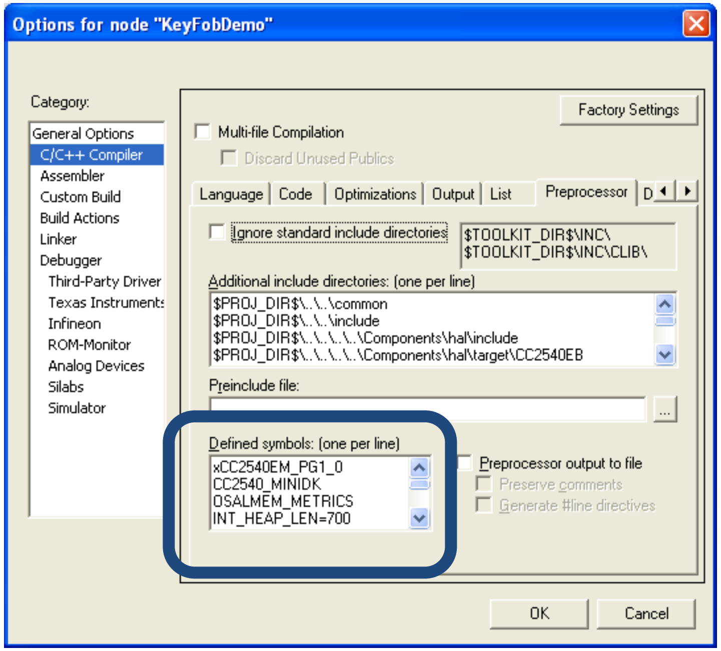
GATT Service Profiles: InitService Function

- InitService function called by application
- When InitService function is called, two variables must be created
  - gattService_t service – includes the number of attributes from the service, and the attribute array itself
  - gattServiceCBs_t serviceCBs – includes two function pointers: ReadAttrCB and ValidateWriteAttrCB (if service doesn't have any readable or writeable attributes, the corresponding pointer can be set to NULL)
- Function calls GATTServApp_RegisterService, with the two variables as parameters to register the attributes and callback functions with the GATT server application

Source Code and Project Notes

- Capital letters at the beginning of a function or variable indicate that it is public or global; lowercase letters indicate private or local

- Build is dependent on having a correct set of preprocessor defined symbols, which can be found in the IAR project options menu



□ Multi-file Compilation
□ Discard Unused Publics

□ Ignore standard include directories

```
$TOOLKIT_DIR$\INC\
$TOOLKIT_DIR$\INC\CLIB\
```

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\common
$PROJ_DIR$\..\..\include
$PROJ_DIR$\..\..\..\..\Components\hal\include
$PROJ_DIR$\..\..\..\..\Components\hal\target\CC2540EB
```

Preinclude file:

Defined symbols: (one per line)

```
xCC2540EM_PG1_0
CC2540_MINIDK
OSALMEM_METRICS
INT_HEAP_LEN=700
```

□ Preprocessor output to file
□ Preserve comments
□ Generate #line directives

GATT Service Profiles: RegisterAppCBs Function

- Only required if profile needs to notify application of information related to the profile

- Examples:
  - In proximity service profile, application needs to know if link-loss or path-loss alert characteristic values have changed
  - In accelerometer profile, application needs to know if accelerometer enabler characteristic value is changed

- Profile must define a type for the callback function pointer in the header file

GATT Service Profiles: Notifications

- Notification / indication handling is typically part of the SetParameter function

- The criteria for when to send notifications or indications can either be set in profile itself or in the application
  - Might be defined by a profile specification