Airoha IoT SDK UI Framework Developers Guide

Version: 1.1

Release date: 12 June 2019

Document Revision History

Revision	Date	Description
1.0	11 January 2019	• Initial release
1.1	12 June 2019	• Added APIs and examples

Table of Contents

Lists of Tables and Figures

1. Overview

The UI framework is a middleware module named ui_shell in the codebase. It helps application developers to design and implement applications. To develop an application, the developer may need to handle the application's state machine, with synchronization inside or between applications according to a pre-defined user scenario. Airoha ui_shell provides the capabilities listed below to fulfill this requirement:

- State (activity) with priority and callback function to process received events
- Finite state machine transition by stack methodology
  - The state machine kernel is called as the Activity Stack
- Inter-application communications
  - Receive messages which are sent from other tasks
- Intra-application communications
  - Process the internal event queue

1.1. System architecture

The system architecture with UI framework is shown in Figure 1.

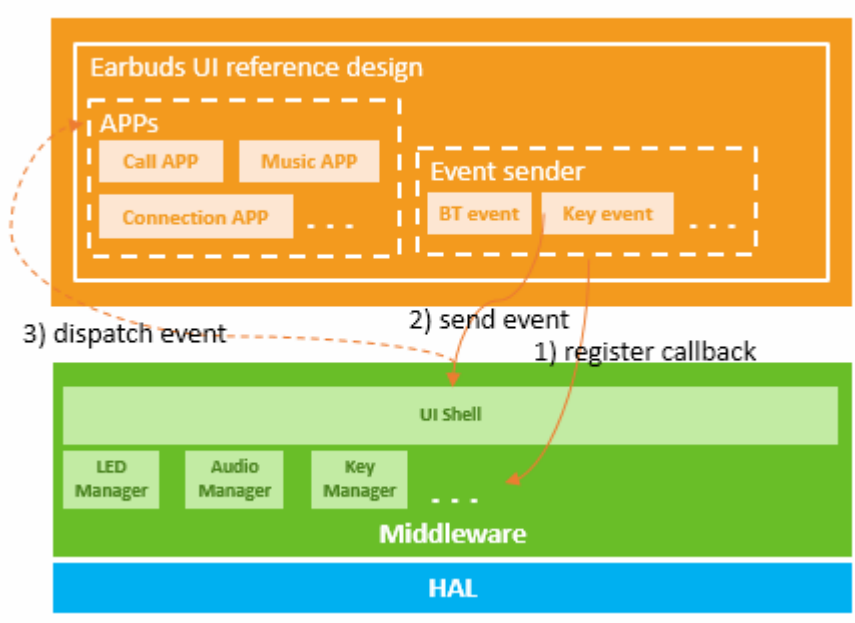

Figure 1. System architecture with UI framework

- UI shell is a module located in middleware
  - Provides an interface that allows the developer to create applications
  - Does not directly call APIs of other middleware or HAL modules
- Applications in project
  - Register proc_event functions which are used in UI shell when it starts
  - Start/stop the UI shell framework

- Create activities and register callbacks which may operate other middleware/HAL modules or send the event messages to UI shell

1.2. Folder structure

The import folders are listed below for your reference. The user can refer to the applications that Airoha has created in the example project. Users who are interested in the detailed design of UI shell can refer to the source code.

UI shell: <MIDDLEWARE_PROPRIETARY>/ui_shell

Application: project/<project path>/src/apps

2. UI Shell Design

2.1. Activity

Activity in UI shell represents a state in the finite state machine of UI shell. Each activity has its own callback function to process received events and decide what to do.

The application can invoke functions to start/stop a new activity. When an activity is created, it is pushed into an activity stack, as shown in Figure 2. A description of the types and priorities of activities, and the activity stack, is provided in the following section.

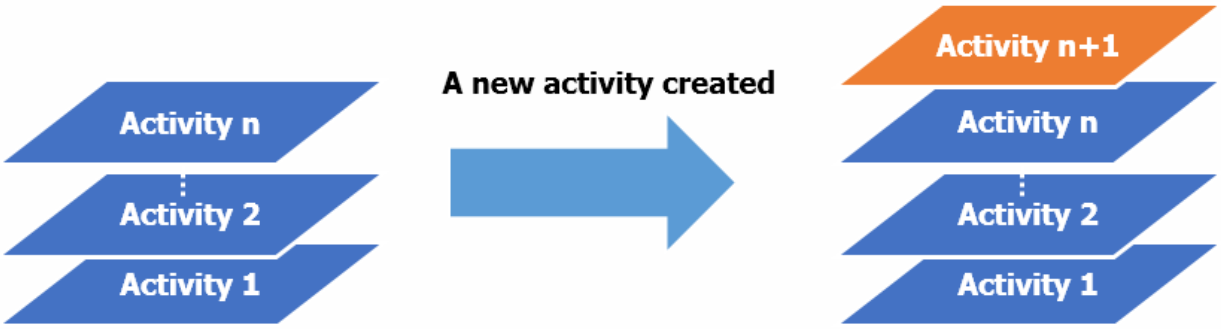

Figure 2. A new activity created

2.1.1. Activity types

There are three different types of activities in UI shell that can be used. The different types of activity are created or used for different purposes.

1) Per-proc activity
   a) A unique background activity for pre-processing. Each project has only one pre-proc activity.
   b) The pre-process activity has the highest priority. Its callback function is executed before all other activities when an event is sent to UI shell.
   c) Life cycle: This activity is never interrupted or disabled, except during the power-off process.
2) Idle activities
   a) Every application can have zero, one, or more than one idle activity.
   b) Idle activities process common messages and start transient activities.
   c) Life cycle: This activity is never interrupted or disabled, except during the power-off process.
3) Transient activities
   a) Activities that are created by idle activities or transient activities when the system is running.
   b) Life cycle: Created/destroyed during a state transition.
   c) Transient activities have a higher priority than idle activities.

2.1.2. Activity Priority

There are seven different priorities in UI shell. The priorities of the activities are in sequence from high to low and are as follows: highest; high; middle; low; lowest; idle_top; and idle_background.

- The highest, high, middle, low, and lowest priorities are used by transient activities.
  - For example, a transient activity for showing the LED of the battery status may be a low priority, and a different transient activity for showing the LED behavior during a search may be a high priority.
- Idle_top and Idle_background are used by idle activities.
  - Usually, there is a unique home screen idle activity using idle_top. Other background activities use idle_background.

2.1.3. Activity stack

Activity stack is a container used to control finite state machine (activities) in UI shell. A project only has one activity which has only one activity stack, and all applications share the same activity stack. The highest priority activities are at the top of the stack and the lowest priority activities are at the bottom of the stack.

Each created activity is put into the activity stack based on its priority. A newly created activity is above other activities with same priority. For example, if there are two applications in the system, the sequence for creating activities is:

- Idle activity 1 and 2 are created when the system starts.
- Idle activity 2 starts transient activity 1.
- Idle activity 1 starts transient activity 2.
- Transient activity 1 starts transient activity 3.

In this situation, the activity stack appears as shown in Figure 3.

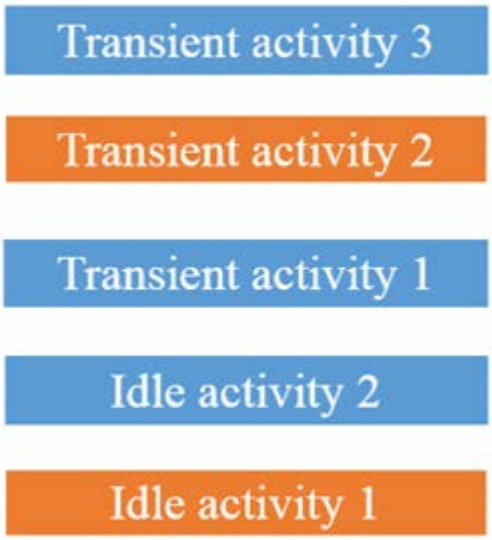

Figure 3. Stack example of two applications

2.2. Event Message

UI shell uses event messages for synchronization. Modules or applications can send events to UI shell. UI shell dispatches the events to activities that already exist in the activity stack. Events can be separate to internal events (pre-defined by UI shell, as described in next section) and user-defined events (i.e. customized by the application developer). Both internal and user-defined events use the same message processing flow.

2.2.1. Process event message

UI shell uses a loop to process events. When an event is sent to UI shell, this event message moves through the activity stack with the same flow as shown in Figure 4.

When the UI shell task starts, it checks the event messages in its internal queue. If there are event messages in the queue, UI shell goes through the activity stack to process the message. If there is no message in the queue, the UI shell task waits until a new message is received.

After a message is processed, UI shell goes back to check the internal queue.

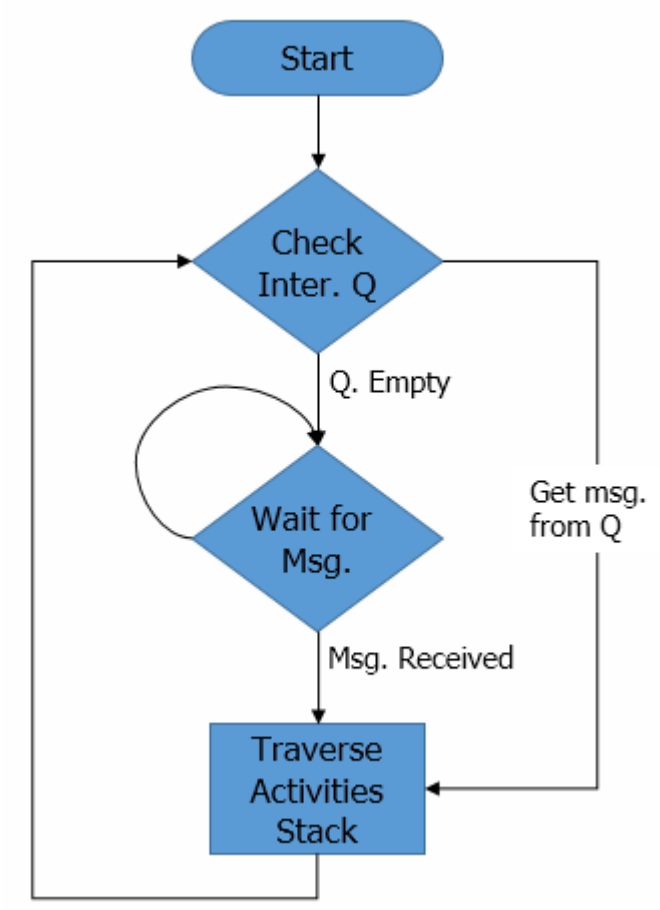

Figure 4. UI shell process event messages

2.2.2. Message traverse

Pre-proc activity first processes the event. Transient and idle activities in the activity stack are processed one message at a time.

An event message is processed by activities from the top to the bottom in the activity stack. The callback function defined in an activity is called when the activity is processing a message. The return value of a callback function decides whether the event message must dispatch to the next activity. A value of “true” means the event must not dispatch to the next activity; a value of “false” means the event is dispatched to the next activity for processing.

2.3. Activity and Event implementation Note

2.3.1. Events

2.3.1.1. Internal events

In the activity stack, if Activity A starts Activity B then, Activity A is Activity B’s “trunk”, and Activity B is Activity A’s “branch”.

When creating or destroying an activity, the activities may receive these events:

- EVENT_ID_SHELL_SYSTEM_ON_CREATE
  - An activity receives this event when it is created.
- EVENT_ID_SHELL_SYSTEM_ON_PAUSE
  - When an activity is created at the top of the stack, the previous top activity receives this event.
- EVENT_ID_SHELL_SYSTEM_ON_RESUME
  - When a top activity is destroyed, the subsequent top activity becomes the top activity and receives this event.
- EVENT_ID_SHELL_SYSTEM_ON_DESTROY
  - When an activity must be destroyed, the activity receives this event before the activity is destroyed.
- EVENT_ID_SHELL_SYSTEM_ON_REFRESH
  - When an activity is created at the top of stack or the second top activity becomes the top activity.
  - This event is received after EVENT_ID_SHELL_SYSTEM_ON_CREATE or when the EVENT_ID_SHELL_SYSTEM_ON_RESUME event processed.
- EVENT_ID_SHELL_SYSTEM_ON_RESULT
  - When a branch activity sends data to its trunk activity, the trunk activity receives this event.
- EVENT_ID_SHELL_SYSTEM_ON_REQUEST_ALLOWANCE
  - When there is an activity request allowance, every activity receives the event. The activity can return true to allow the request.
- EVENT_ID_SHELL_SYSTEM_ON_ALLOW
  - When an allowance request is allowed by all active activities, the activity which sends the request receives this event.

2.3.1.2. User defined events

Refer to project/<project path>/inc/apps/events/apps_events_event_group.h and project/<project path>/src/apps/events for the user defined events.

2.3.2. Exchange data between activities

When Activity A creates another Activity B, the developer may want to have information returned from Activity B. To decrease the possibility of using global variables, we strongly suggest that the customer uses extra data to transfer the pointer, value, or any other generated data to different activities.

- The branch activity sends the result to the trunk activity
  - The branch activity can call the set_result() function to notify the trunk activity.
  - UI shell sends EVENT_ID_SHELL_SYSTEM_ON_RESULT to the trunk activity.

2.3.3. Getting allowance from all activities

Sometimes, one activity must ask all other activities for permission to do something. For example, one activity should ask all other activities if it can execute the power off sequence at a specific time. When an activity receives the request for permission, it can return “true” to immediately grant that permission. Otherwise, it can return “false” to temporarily disallow the request, and then later call a function to grant it permission.

The steps are as follows:

1) An activity sends a request for allowance.
2) All activities receive the request event.
   a) Activities receive the event: EVENT_ID_SHELL_SYSTEM_ON_REQUEST_ALLOWANCE
3) If an activity returns “true”, that specific activity gives the allowance.
4) If an activity returns “false”, it can call the function again at a later time to give the allowance.
5) If all activities allow the request, the activity which sent the request receives the EVENT_ID_SHELL_SYSTEM_ON_ALLOW event.

2.4. APIs

A definition of each API in UI shell is provided in the following section. Note that the detailed version of the API description is also provided in the API reference and header file of UI shell.

2.4.1. UI Shell

2.4.1.1. ui_shell_status_t ui_shell_start(void)

Start the UI shell framework.

2.4.1.2. ui_shell_status_t ui_shell_finish(void)

Stop the UI shell framework and destroy any temporary data used in UI shell.

2.4.1.3. ui_shell_status_t ui_shell_set_pre_proc_func(ui_shell_proc_event_func_t proc_event)

Set the proc_event function of the pre-proc activity. The function should be called before UI shell starts.

2.4.2. Activity

2.4.2.1. ui_shell_status_t ui_shell_start_activity(ui_shell_activity_t *self, ui_shell_proc_event_func_t proc_event, ui_shell_activity_priority_t priority, void *extra_data, size_t data_len)

Start an activity. The proc_event is the callback function for processing the received events.

2.4.2.2. ui_shell_status_t ui_shell_finish_activity(ui_shell_activity_t *self, ui_shell_activity_t *target_activity)

Destroy an existing activity.

2.4.2.3. ui_shell_status_t ui_shell_back_to_idle(ui_shell_activity_t *self)

Destroy all transient activities. The Idle top activity becomes the top activity in the activity stack.

2.4.2.4. ui_shell_status_t ui_shell_set_result(ui_shell_activity_t *self, void *data, size_t data_len)

Return data to an activity which starts the current activity.

2.4.2.5. ui_shell_status_t ui_shell_refresh_activity(ui_shell_activity_t *self, ui_shell_activity_t *target)

Request UI shell to send an ON_REFRESH event to the target activity.

2.4.3. Event

2.4.3.1. ui_shell_status_t ui_shell_send_event(bool from_isr, ui_shell_event_priority_t priority, uint32_t event_group, uint32_t event_id, void *data, size_t data_len, void (*special_free_extra_func)(void), uint32_t delay_ms)

Send an event to UI shell. UI shell dispatches it to the activities after a delay.

2.4.3.2. ui_shell_status_t ui_shell_remove_event(uint32_t event_group, uint32_t event_id)

Remove all unprocessed events which match the event group and the event id from the event list.

2.4.4. Allowance

2.4.4.1. ui_shell_status_t ui_shell_request_allowance(ui_shell_activity_t *self, uint32_t request_id)

Send an allowance request to UI shell. UI shell sends it to all active activities. The activities can return true to immediately allow it, or use ui_shell_grant_allowance() to allow it at a later time.

2.4.4.2. ui_shell_status_t ui_shell_grant_allowance(ui_shell_activity_t *self, uint32_t request_id)

If an activity did not allow the request temporarily when it received EVENT_ID_SHELL_SYSTEM_ON_GET_ALLOWN, it can call the function when it allows the request at a later time.

2.5. Examples

The following two examples show how to use UI shell. For more examples of the implementation, refer to the example project in the SDK.

2.5.1.1. Scenario 1

When the user receives an incoming call and they press a key to accept the call, the procedure is as follows:

1) The BT module calls a BT callback function which is defined in an event sender.
2) The event sender sends an event with the incoming call information to UI shell by calling the ui_shell_send_event() API.
3) UI shell dispatches the incoming call event to all activities.
4) When a CALL APP idle activity receives the event in its pre-proc function, it starts CALL APP_transient activity by calling API ui_shell_start_activity().
5) UI shell starts calling the transient activity.
6) When the user presses a key to accept the call, the key module calls a key callback function which is defined in an event sender.
7) The event sender sends an event with the key press information to UI shell by calling the ui_shell_send_event() API.
8) UI shell dispatches the event to every activity. Because the call transient activity has a higher priority than the other activities, it receives the event before the other activities.
9) CALL APP transient activity receives the event and call APIs in the BT module to accept the incoming call. It returns true because the key event should not dispatch to the next activity.

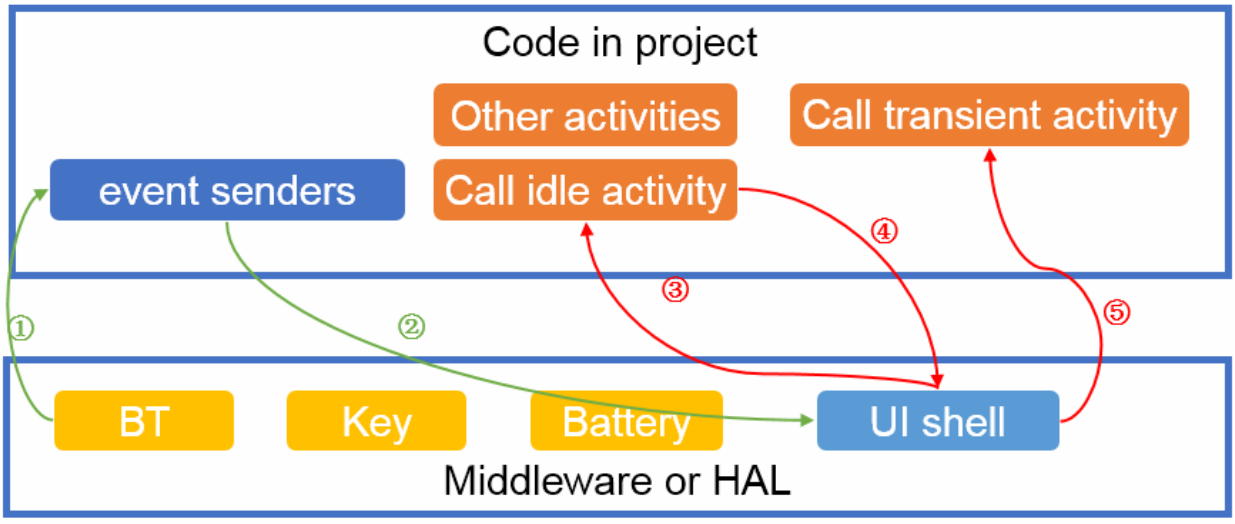

Figure 5. Receive incoming call event

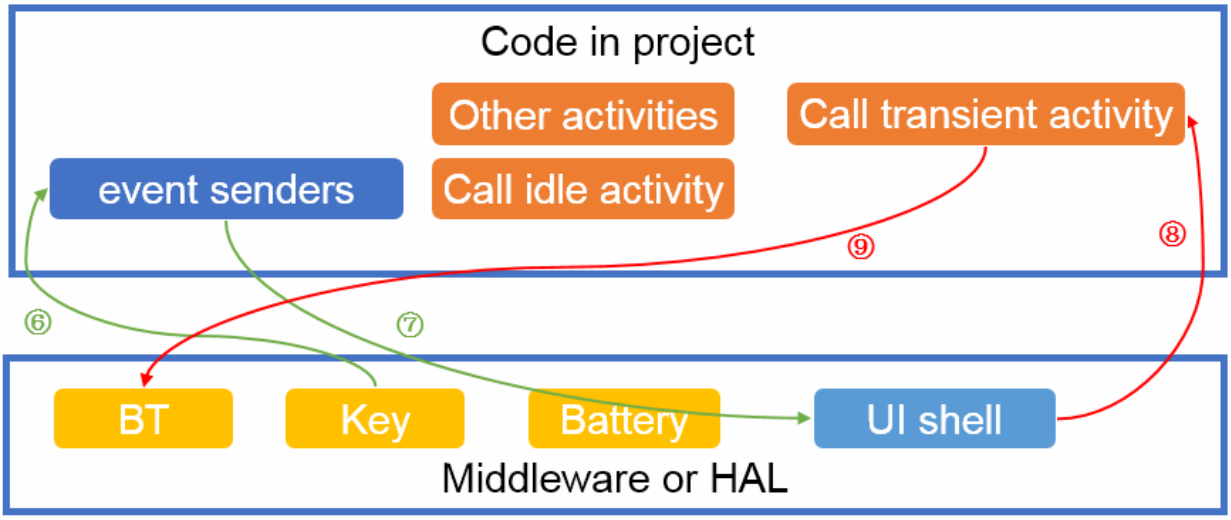

Figure 6. Receive key event to accept call

2.5.1.2. Scenario 2

When the battery is too low to do the power off, the procedure is as follows:

1) The battery module calls a battery callback function which is defined in an event sender.
2) The event sender sends an event with battery voltage information to UI shell by calling API ui_shell_send_event().
3) UI shell dispatches the event to every activity.
4) When BATTERY APP idle activity receives the event, it sends another event with a power off request to UI shell by calling ui_shell_send_event().
5) After the battery voltage information event is processed by the complete activity stack, UI shell dispatches the power off request event to every activity.
6) HOMESCREEN APP idle activity receives the event. It calls the API in HAL to do the power off.

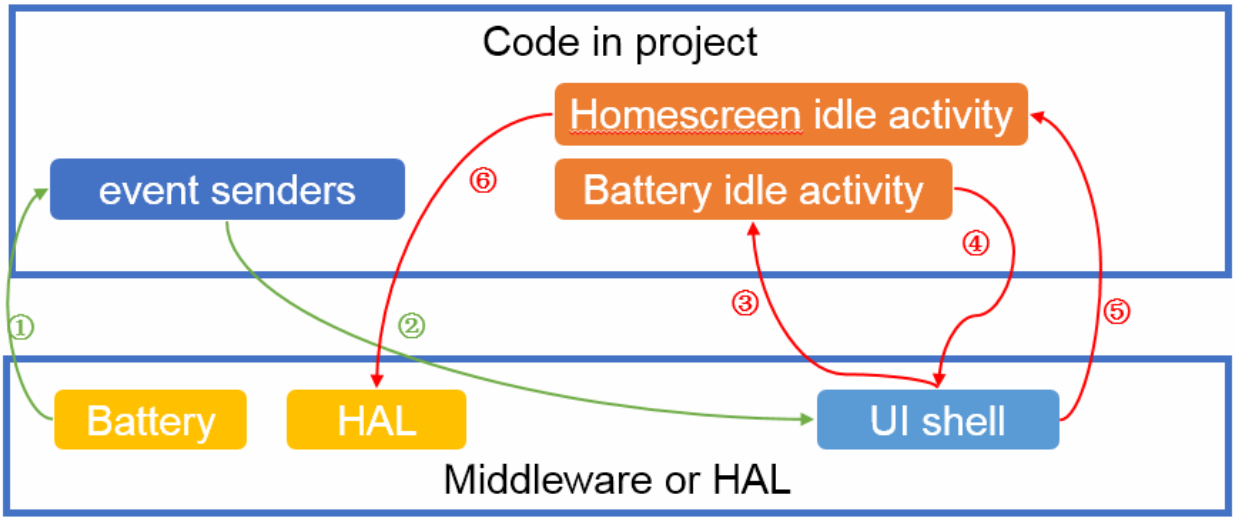

Figure 7. Power off when low battery