Application note Hue EDK connection flow

Connecting to a bridge

Main

The main way of connecting to a bridge is using `ConnectBridgeAsync()`. The behavior is as follows:

- If a valid bridge is already configured, it will connect to that bridge
- If no valid bridge is configured yet, it will first search for bridges on the network and connect if a bridge is found

In the background

If instead of the user explicitly requesting to connect to a bridge, the application wants to discover bridges in the background, there is the option to use `ConnectBridgeBackgroundAsync()`. This does exactly the same as `ConnectBridgeAsync()` except for the for the following differences:

- The search strategy does not include a brute force retry if the first search cycle didn't deliver any results
- Bridge models without streaming capability will be ignored
- When a new bridge is found, instead of immediately start scanning for the user pressing the pushlink button, it will just finish and remember the discovered bridge. When the application is ready to show the UI to the user, `ConnectBridgeAsync()` can be called to 'resume'.

Manual IP

In case the bridge discovery doesn't work even though the user has a bridge on the same network, there is a last resort in which the user enters the bridge ip address manually. In this case `ConnectBridgeManualIpAsync(const string &ipAddress)` can be called which skips both loading of any already configured bridge and new bridge discovery. Instead it will immediately try to connect to the bridge on the given ip address. The rest of the procedure is the same as `ConnectBridgeAsync()`.

Getting feedback

Via asynchronous callback

Before connecting, the application can use `RegisterFeedbackCallback(FeedbackMessageCallback callback)` to get feedback during the connection procedure via a callback. The callback provides a `FeedbackMessage` which indicates:

- The type of ongoing request: `GetRequestType()`
- An enum identifier: `GetId()`
- A tag (or 'string id') used for looking up the user message translation: `GetTag()`
- A message type: `GetType()`, can be `USER` (needs user interaction) or `INFO`
- If the message type is USER, a user message string explaining the user what to do in the language the EDK is configured in: `GetUserMessage()`
- A debug message string: `GetDebugMessage()`

Note that instead of a callback, the application can choose to implement the `IFeedbackMessageHandler` interface and set it via `RegisterFeedbackHandler(FeedbackMessageHandlerPtr handler)`.

Via synchronous request

By calling `GetConnectionResult()` the application can request the result of the last connection procedure at any time. If the result is `Completed` or `ActionRequired`, the procedure has delivered a bridge object. In case of `ActionRequired`, the type of required action can be found via `GetLoadedBridge()->GetStatus()` and the user can be informed what to do via `GetLoadedBridge()->GetUserStatus()`.

Handling the different results

After starting a bridge connection procedure, you can end up in five different situations in terms of what action is needed. Below table describes these. It assumes using `ConnectBridgeAsync()` but would be very similar for the other connection methods.

Situation	**FeedbackMessage id**	**Suggested action**
No bridge found	`NO_BRIDGE_FOUND`	In case the search was started automatically in the background: • No action required In case the search was requested manually by the user (e.g. via settings menu): • Show the user message. • Offer a retry option, calling `ConnectBridgeAsync()` again. • A way to skip.
New bridge found	`PRESS_PUSH_LINK`	• Show user message together with an image of the bridge to guide the user to press the button on the bridge. • Offer an option to not enable the Hue integration. In this case call `AbortConnecting()`. • If a user doesn't want to use the integration, this preference should probably be saved to not automatically search again next time the user starts the game. Possibly inform the user how to still connect manually later.
User needs to select which entertainment setup to use (in application)	`SELECT_GROUP`	• Show the user message. • Show the list of available groups via `GetLoadedBridge()->GetGroups()` and using `GetName()`. • When the user selects a group call `SelectGroupAsync(GroupPtr group)`.

User needs to take action external of application (e.g. create entertainment setup or upgrade bridge using the mobile Hue app)	`BRIDGE_NOT_FOUND`
	`INVALID_MODEL`
	`INVALID_VERSION`
	`NO_GROUP_AVAILABLE`
	`BUSY_STREAMING`
	`(INTERNAL_ERROR)`
		• Show the user message.
		• Offer a button to retry, calling `ConnectBridgeAsync()` again.
		• A way to skip.

Connected to bridge	`DONE_COMPLETED`	• Possibly show a small connect confirmation.

When the connection flow is in progress

When the user has actively requested to connect to a bridge, the UI should probably show that the connection procedure is in progress and offer a way to cancel.

Only one connection procedure can be active at any time. For any procedure it is always guaranteed that if it is started `PROCEDURE_STARTED` is called and if it is finished (whether successful or not) `PROCEDURE_FINISHED` is called. Similarly when the procedure is ongoing a call to `GetConnectionResult()` will return `Busy`.

This can be used to e.g. show a spinner and/or change the text of the connect button into a cancel button. To cancel call `AbortConnecting()`.

Obviously, this is only relevant if there is any UI, not if the application is discovering or connecting to bridges in the background hidden for the user.

Resetting the bridge configuration

A user may want to connect to another bridge than the one which is currently configured. For this there needs to be a way for the user to reset the current bridge configuration. This can be done by just calling `ResetBridgeInfoAsync()`.