Dialogue API Reference

This API contains two types of messages:

- **Inbound Messages** are produced by the platform, you should "subscribe" and react to them
- **Outbound Messages** are sent to the platform, you should "publish" them and the platform will react accordingly

---

Intent

Inbound Message

MQTT Topic

You should subscribe to `hermes/intent/<intentName>`.

Replace `<intentName>` by the name of the intent you want to handle. You can use the MQTT wildcard `#` to receive all intents.

Android Listener

You should register a listener using `setOnIntentDetectedListener` on `SnipsPlatformClient`.

iOS Closure

You can write your closure in the `onIntentDetected` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onIntentDetected = { intent in
    // Your code here
}
```

This is the main message the handler code should subscribe to. It is sent by the Dialogue Manager when an intent has been detected.

Note that it is the handler's responsibility to inform the Dialogue Manager of what it should do with the current session by sending either a Continue Session or an End Session with the current `sessionId`

Payload

Key	Value
`sessionId`	*String* - Session of the intent detection. The client code must use it to continue or end the session.
`customData`	*Optional String* - Custom data provided in the Start Session or a Continue Session.
`siteId`	*String* - Site where the user interaction took place.
`input`	*String* - The user input that has generated this intent.
`intent`	*Object* - Structured description of the intent classification, see Intent Classification below.
`slots`	*Optional Array of Objects* - Structured description of the detected slots for this intent if any, see Slot below.
`asrTokens`	*Optional Double Array of Objects* - Structured description of the tokens the ASR captured on for this intent. The first level of arrays represents each invocation of the ASR, the second one are the tokens captured, see ASR Token below. Note that this is not mapped on Android and iOS yet
`asrConfidence`	*Optional Number* - ASR confidence score between 0 and 1, 1 being sure.

`alternatives`	*Optional Array of Objects* - Structured description of alternatives intent resolution see Intent Alternatives below

Intent Classification

Key	Value
`intentName`	*String* - The name of the detected intent.
`confidenceScore`	*Number* - The probability of the detection, between 0 and 1, 1 being sure.

Slot

Key	Value
`confidence`	*Number* - Confidence of the slot, between 0 and 1, 1 being confident.
`raw_value`	*String* - The raw value of the slot, as is was in the input.
`value`	*String* - The resolved value of the slot.
`entity`	*String* - The entity of the slot.
`slotName`	*String* - The name of the slot.
`range`	*Optional Object* - The range where the slot can be found in the input. The object contains 2 *integer* fields: `start` representing the beginning of the range (inclusive) and `end` representing the end (exclusive).

ASR Token

Key	Value
`value`	*String* - The value of the token
`confidence`	*Number* - Confidence of the token, between 0 and 1, 1 being confident
`rangeStart`	*Integer* - The start range in which the token is in the original input
`rangeEnd`	*Integer* -The end range in which the token is in the original input
`time`	*Object* - Time when this token was detected. The object contains 2 *Number* fields: `start` representing the start time and `end` representing the end time

Intent Alternative

Key	Value
`intentName`	*String* - The value of the token
`confidenceScore`	*Number* - Confidence of the token, between 0 and 1, 1 being confident
`slots`	*Optional Array of Objects* - Structured description of the detected slots for this alternative intent if any, see Slot below.

Start Session

Outbound Message

MQTT Topic

You should publish to `hermes/dialogueManager/startSession`.

Android Method

You should send a message using `dialogueStartSession` on `SnipsPlatformClient`.

iOS Method

Three methods are at your disposal to start a session:

```
func startSession() throws
func startSession(message: StartSessionMessage) throws
func startSession(text: String? = nil, intentFilter: [String]? = nil, canBeEn

// Usage
let snips = SnipsPlatform(...)
try? snips.startSession()
```

You can send this message to programmatically initiate a new session. The Dialogue Manager will start the session by asking the TTS to say the text (if any) and wait for the answer of the end user.

Payload

Key	Value
`siteId`	*Optional String* - Site where to start the session
`init`	*Object* - Session initialization description: Action or Notification. See below
`customData`	*Optional String* - Additional information that can be provided by the handler. Each message related to the new session - sent by the Dialogue Manager - will contain this data

Session Initialization: Action

Use this type when you need the end user to respond

Key	Value
`type`	*"action"*
`text`	*Optional String* - Text that the TTS should say at the beginning of the session.

`canBeEnqueued`	*Boolean* - if true, the session will start when there is no pending one on this siteId, if false, the session is just dropped if there is running one.
`intentFilter`	*Optional Array of Strings* - A list of intents names to restrict the NLU resolution on the first query.
`sendIntentNotRecognized`	*Optional Boolean* - Indicates whether the dialogue manager should handle non recognized intents by itself or sent them as an Intent Not Recognized for the client to handle. This setting applies only to the next conversation turn. The default value is false (and the dialogue manager will handle non recognized intents by itself)

Session Initialization: Notification

Use this type when you only want to inform the user of something without expecting a response.

Key	Value
`type`	*"notification"*
`text`	*String* - Text the TTS should say

Session Queued

Inbound Message

MQTT Topic

You should subscribe to `hermes/dialogueManager/sessionQueued`.

Android Listener

You should register a listener using `setOnSessionQueuedListener` on `SnipsPlatformClient`.

iOS Closure

You can write your closure in the `onSessionQueuedHandler` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onSessionQueuedHandler = { message in
    // Your code here
}
```

This message is sent by the Dialogue Manager when it receives a Start Session message and the corresponding site is busy. Only Start Session messages with a notification initialisation or an action initialisation with the `canBeEnqueued` flag set to true can be enqueued. When the site is free again, this session will be started.

Payload

Key	Value
`sessionId`	*String* - Session identifier that was enqueued.
`siteId`	*String* - Site where the user interaction will take place.
`customData`	*Optional String* - Custom data provided in the Start Session .

Session Started

Inbound Message

MQTT Topic

You should subscribe to `hermes/dialogueManager/sessionStarted` .

Android Listener

You should register a listener using `setOnSessionStartedListener` on `SnipsPlatformClient`.

iOS Closure

You can write your closure in the `onSessionStartedHandler` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onSessionStartedHandler = { message in
    // Your code here
}
```

This message is sent by the Dialogue Manager when a new session is started, either following a wakeword or programmatically.

Payload

Key	Value
`sessionId`	*String* - Session identifier that was started.
`siteId`	*String* - Site where the user interaction is taking place.
`customData`	*Optional String* - Custom data provided in the Start Session .

Continue Session

Outbound Message

MQTT Topic

You should publish to `hermes/dialogueManager/continueSession`.

Android Method

You should send a message using `dialogueContinueSession` on `SnipsPlatformClient`.

iOS Method

Two methods are at your disposal to continue a session:

```
func continueSession(sessionId: String, text: String, intentFilter: [String]?
func continueSession(message: ContinueSessionMessage) throws

// Usage
let snips = SnipsPlatform(...)
try? snips.continueSession(sessionId: "xxxx", text: "Next command")
```

You should send this after receiving received an Intent when you want to continue the session for example to ask additional information to the end user.

Be sure to use the same `sessionId` as the one in the Intent message.

Payload

Key	Value
`sessionId`	*String* - Identifier of the session to continue.
`text`	*String* - The text the TTS should say to start this additional request of the session.

`intentFilter`	*Optional Array of String* - A list of intents names to restrict the NLU resolution on the answer of this query. If this contains unknown intent names, the NLU will post an error message and the session will be closed.
`customData`	*Optional String* - an update to the session's custom data. If not provided, the custom data will stay the same.
`sendIntentNotRecognized`	*Optional Boolean* - Indicates whether the dialogue manager should handle non recognized intents by itself or sent them as an Intent Not Recognized for the client to handle. This setting applies only to the next conversation turn. The default value is false (and the dialogue manager will handle non recognized intents by itself).
`slot`	*Optional String*, requires `intentFilter` to contain a single value - If set, the dialogue engine will not run the the intent classification on the user response and go straight to slot filling, assuming the intent is the one passed in the `intentFilter`, and searching the value of the given slot

End Session

Outbound Message

MQTT Topic

You should publish to `hermes/dialogueManager/endSession`.

Android Method

You should send a message using `dialogueEndSession` on `SnipsPlatformClient`.

iOS Method

Two methods are at your disposal to end a session:

```
func endSession(sessionId: String, text: String? = nil) throws
func endSession(message: EndSessionMessage) throws

// Usage
let snips = SnipsPlatform(...)
try? snips.endSession(sessionId: "xxxx")
```

You should send this after receiving received an Intent when you want to end the session.

Be sure to use the same `sessionId` as the one in the Intent message.

Payload

Key	Value
`sessionId`	*String* - Identifier of the session to end.
`text`	*Optional String* - The text the TTS should say to end the session.

If the text is null, the Dialog Manager will immediately send a Session Ended after receiving this message, otherwise, the Session Ended will be sent after the text is said.

Session Ended

Inbound Message

MQTT Topic

You should subscribe to `hermes/dialogueManager/sessionEnded` .

Android Listener

You should register a listener using `setOnSessionEndedListener` on `SnipsPlatformClient`.

iOS Closure

You can write your closure in the `onSessionEndedHandler` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onSessionEndedHandler = { message in
    // Your code here
}
```

This message is sent by the Dialogue Manager when a session is ended.

Payload

Key	Value
`sessionId`	*String* - Session identifier of the ended session.
`customData`	*Optional String* - Custom data provided in the Start Session or a Continue Session.
`siteId`	*String* - Site where the user interaction took place.
`termination`	*Object* - Structured description of why the session has been ended. See below.

Session Termination Type

Key	Value

`reason`

String - the reason why the session was ended this can have the following values:

- `nominal` : the session ended as expected (a `endSession` message was received).
- `abortedByUser` : the session aborted by the user.
- `intentNotRecognized` : the session ended because no intent was successfully detected.
- `timeout` : The session timed out because no response from one of the components or no Continue Session or End Session from the handler code was received in a timely manner
- `error` : The session failed with an error.

Inbound Message

MQTT Topic

You should subscribe to `hermes/dialogueManager/intentNotRecognized` .

Android Listener

You should register a listener using `onIntentNotRecognizedListener` on SnipsPlatformClient.

iOS Closure

You can write your closure in the `onIntentNotRecognizedHandler` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onIntentNotRecognizedHandler = { message in
    // Your code here
}
```

This message is sent by the dialogue manager when it failed to recognize and intent AND you requested the dialogue manager to notify you of such events using the `sendIntentNotRecognized` flag in the last

Start Session or Continue Session

Payload

Key	Value
`sessionId`	*String* - Session identifier of the session that generated this intent not recognized event.
`customData`	*Optional String* - Custom data provided in the Start Session or a Continue Session.
`siteId`	*String* - Site where the user interaction took place.
`input`	*Optional String* - The input, if any that generated this event.

Configure

Outbound Message

MQTT Topic

You should publish to `hermes/dialogueManager/configure`.

Android Method

This feature is not available yet on Android.

iOS Method

You can send this message to programmatically configure the scope of intents that are enabled at a given time. By default, all intents are enabled by default unless specified otherwise in the console. Refers to the dedicated Console tutorial for further details. The behaviour implemented by the `intentFilter`

attributes for the start session and continue session locally overwrites the set of enabled/disabled intents **only** for the next user turn.

Payload

Key	Value
`siteId`	*Optional String* - Site where to start the session
`intents`	*Array of Intent configuration objects* - see Intent configuration description below

Intent configuration

Key	Value
`intentId`	*String* - Intent name to configure
`enable`	*Boolean* - if true, the intent will be enabled which means that the Intent message for the related `intent_name` is enabled and will be triggered if the intent is detected.

Entities Injection

Outbound Message

MQTT Topic

You should publish to `hermes/injection/perform` .

Android Method

You should send a `InjectionRequestMessage` message using `requestInjection` on `SnipsPlatformClient`.

```
fun SnipsPlatformClient.requestInjection(injectionRequestMessage: InjectionRe

// Usage
val snipsPlatformClient = SnipsPlatformClient.Builder(...)
val operations = listOf(InjectionOperation(InjectionKind.AddFromVanilla, muta
val lexicon = mutableMapOf<String, List<String>>()
val request = InjectionRequestMessage(operations, lexicon, null, null)

snipsPlatformClient.requestInjection(request)
```

iOS Method

```
func requestInjection(with message: InjectionRequestMessage) throws

// Usage
let snips = SnipsPlatform(...)
let operation = InjectionRequestOperation(entities: ["locality": ["wonderland"
try! snips.requestInjection(with: InjectionRequestMessage(operations: [operati
```

Entities injection allows you to update both the ASR and the NLU models directly on the device. Each intent within an assistant may contain some slots, and each slot has a specific type that we call an *entity*. If you have a *contact_name* entity that contains a list of contacts in an address book, **Entities Injection** lets you add new contact names to this list.

Once the injection has completed successfuly, there will be an injectionComplete message sent.

ⓘ For a more in-depth explanation of how injection works, check the documentation

Available operations

Two injection operations are currently supported: `add` and `addFromVanilla`. Other operations are under development.

- `add` adds the list of values that you provide to the existing entity values.
- `addFromVanilla` removes all the previously injected values to the entity, and then, adds the list of values provided. Note that the entity values coming from the console will be kept.

Let's illustrate this with an entity having two values: `one` and `two`. Here is how the entity values will be affected after performing some injection operations:

OperationKind	Values to inject	Supported entity values
		one, two
add	three	one, two, three
add	four	one, two, three, four
addFromVanilla	five	one, two, five
add	six	one, two, five, six

The entity values to inject are specified in a JSON file which must respect the following format:

- A key `operations` mapping to a list of operations
- Each operation is a tuple (`OperationKind` , `OperationData`)
- `OperationKind` is the type of injections to perform. See the *Available operations* section for the allowed operations.
- `OperationData` is a dictionary mapping an entity name to a list of values (strings).

For instance, if you are willing to add "The Wolf of Wall Street" to your list of films, just write the following file:

Payload

Key	Type	Description
`id`	*Optional String*	Request identifier for the request
`crossLanguage`	*Optional String*	Language for cross-language G2P

`lexicon`	*Optional Array of (Value, Array of Pronunciation)*	List of pre-computed prononciations to add in a model
`operations`	*Array of (InjectionKind, Array of (Entity, Array of EntityValue))*	List of pre-computed prononciations to add in a model

Injection Complete

Inbound Message

MQTT Topic

You should publish to `hermes/injection/complete`.

Android Method

You should register a listener using `setOnInjectionComplete` on `SnipsPlatformClient`.

iOS Closure

You can write your closure in the `onInjectionComplete` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onInjectionComplete = { message in
    // Your code here
}
```

Once the injection request has been processed, the ASR and NLU are reloaded. Once reloaded, Snips Platform will post on this route.

Payload

Key	Type	Description
`requestId`	*Optional String*	Request identifier for the request

Injection Reset

Outbound Message

MQTT Topic

You should publish to `hermes/injection/reset/perform`.

Android Method

You should send a `InjectionResetRequestMessage` message using `requestInjectionReset` on `SnipsPlatformClient`.

```
fun SnipsPlatformClient.requestInjectionReset(injectionResetRequestMessage: Inj
```

iOS Method

```
1 func requestInjectionReset(with message: InjectionResetRequestMessage = Injec
2
3 // Usage
4 let snips = SnipsPlatform(...)
5 snips.requestInjectionReset()
```

Injection reset will delete previously injected entities and reboot the ASR & NLU. You don't need to relaunch the platform. Once the injection is complete, it will post a message on hermes.

Payload

Key	Type	Description
`requestId`	*Optional String*	Request identifier for the request

Injection Reset Complete

Inbound Message

MQTT Topic

You should publish to `hermes/injection/reset/complete`.

Android Method

You should register a listener using `setOnInjectionResetComplete` on `SnipsPlatformClient`.

iOS Closure

You can write your closure in the `onInjectionResetComplete` property on `SnipsPlatform` client.

```
let snips = SnipsPlatform(...)
snips.onInjectionResetComplete = { request in
    // Your code here
}
```

Payload

Key	Type	Description

`requestId`	*Optional String*	Request identifier for the request