

ORACLE

# Webhook channel

---

# Program agenda

---

- 1 About webhook channel
- 2 Creating a webhook channel
- 3 Building a webhook implementation

## About the webhook channel

---

” A webhook is required for channels not natively supported by the Oracle Digital Assistant

A webhook channel provides a direct connection into digital assistants or skills

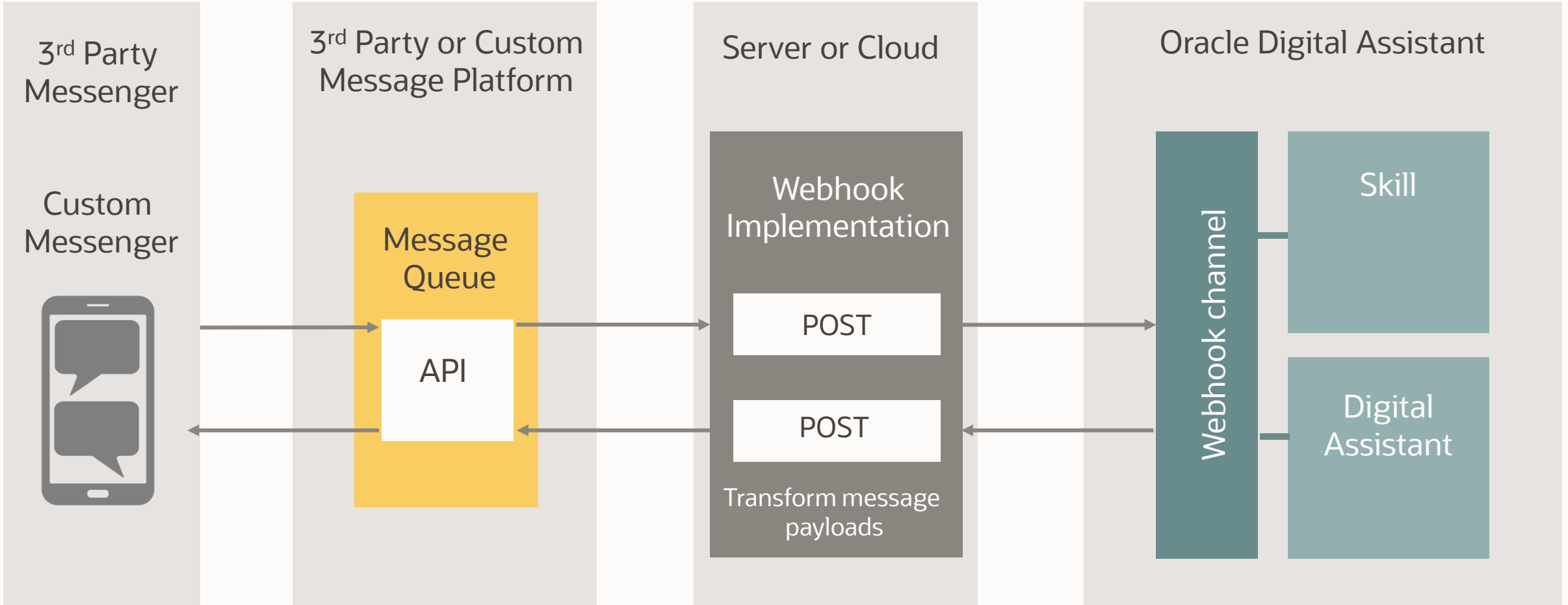
Webhook acts as an intermediary that handles and tracks user conversations

Transforms message payloads

Provides a message server

*E.g.* Alexa, Google Home or Telegram integration

# Conversation architecture



# Program agenda

---

- 1 About webhook channel
- 2 **Creating a webhook channel**
- 3 Building a webhook implementation

# Creating a webhook

---

Provide a publicly accessible HTTP messaging server

- Relays messages between the user device and digital assistant (or skill)

Create a webhook with

- A POST call that enables the server to receive messages from digital assistant
- A POST call that enables the server to send messages to digital assistant

Provide an URI of the webhook call that receives digital assistant messages

- Needs to be configured on the webhook channel in Oracle Digital Assistant

Obtain the URL that's generated by Oracle Digital Assistant webhook channel

- Provides information how to call into a digital assistant or skill

# Creating a webhook channel in ODA

Choose Webhook as the channel type

Set Payload Version to Conversation Model

Register your webhook endpoint for incoming messages to the Outgoing Webhook URI field

Digital Assistant generates

- the webhook URL for incoming messages
- A secret Key for encrypting messages

Use the Route To selector to associate the channel with a digital assistant or skill

The image shows two overlapping screenshots of the 'Create Channel' dialog in Oracle Digital Assistant. The top screenshot shows the initial configuration with the following fields: Name (WebhookStarter), Description (Optional short description for this channel), Channel Type (Webhook), Payload Version (Conversation Model), Outgoing Webhook URI (https://myendpoint.com), and Session Expiration (minutes) (1,440). The bottom screenshot shows the 'WebhookStarter' channel details, including: Channel Enabled (checked), Route To (Financial.DigitalAssistant - Installed Digital Assistant - 1.1 - 22.00), Name (WebhookStarter), Description (Optional short description for this channel), Channel Type (Webhook), Payload Version (Conversation Model), Outgoing Webhook URI (https://myendpoint.com/bot/receivemessage), Secret Key (L1Z0cYqydQLt2X3HLoVy1ONIHM5ZcaK), Webhook URL (https://oda-cd25acca080449f39f3d28f1d3ffb801-da14.data.digitalassistant.oci.oraclecloud.com/conn...), and Session Expiration (minutes) (1,440). A red circle with a white 'O' is visible in the bottom right corner of the page.

# Program agenda

---

- 1 About webhook channel
- 2 Creating a webhook channel
- 3 Building a webhook implementation



# Building your webhook implementation

The webhook channel configuration gives you

- A URL to call into Oracle Digital Assistant
- A secret key to protect messages

Oracle Digital Assistant's Node.js SDK

- <https://github.com/oracle/bots-node-sdk>
- Client library simplifies the setting up of sending and receiving messages
  - Provides support for message transformation
- Sets the X-Hub-Signature header containing the SHA256 value of the payload

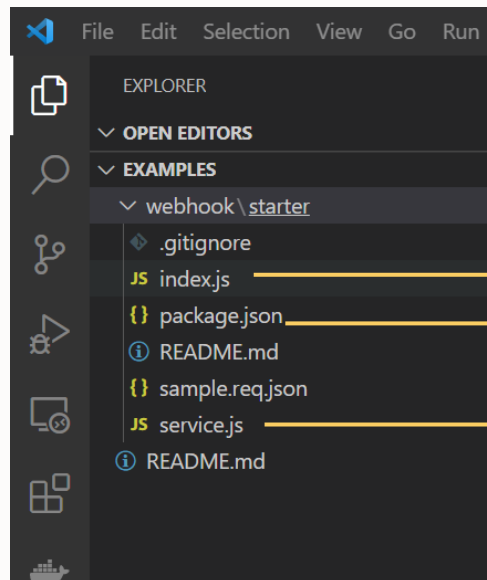
```
{
  "userId": "22343248763458761287",
  "messagePayload": {
    "type": "text",
    "text": "What do you want to do?",
    "actions": [
      {
        "type": "postback",
        "label": "Order Pizza",
        "postback": {
          "state": "askAction",
          "action": "orderPizza"
        }
      },
      {
        "type": "postback",
        "label": "Cancel A Previous Order",
        "postback": {
          "state": "askAction",
          "action": "cancelOrder"
        }
      }
    ]
  }
}
```

# Getting started with your webhook development

The bots-node-sdk is available on:

<https://github.com/oracle/bots-node-sdk>

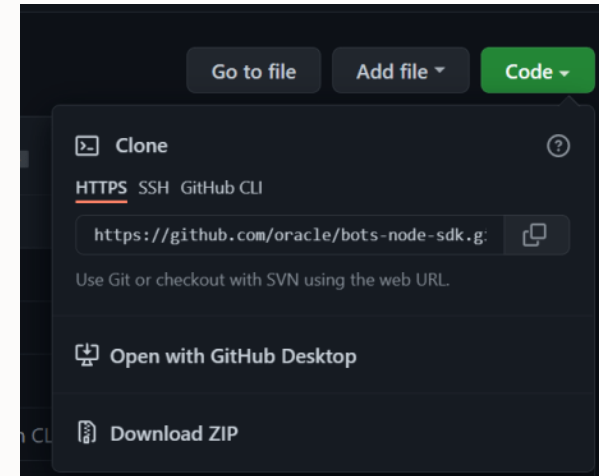
There is a starter webhook folder under **bots-node-sdk-master\examples\webhook\starter**



Node Express server

Node dependencies

Webhook client



# Getting started with your webhook development

Open a terminal window

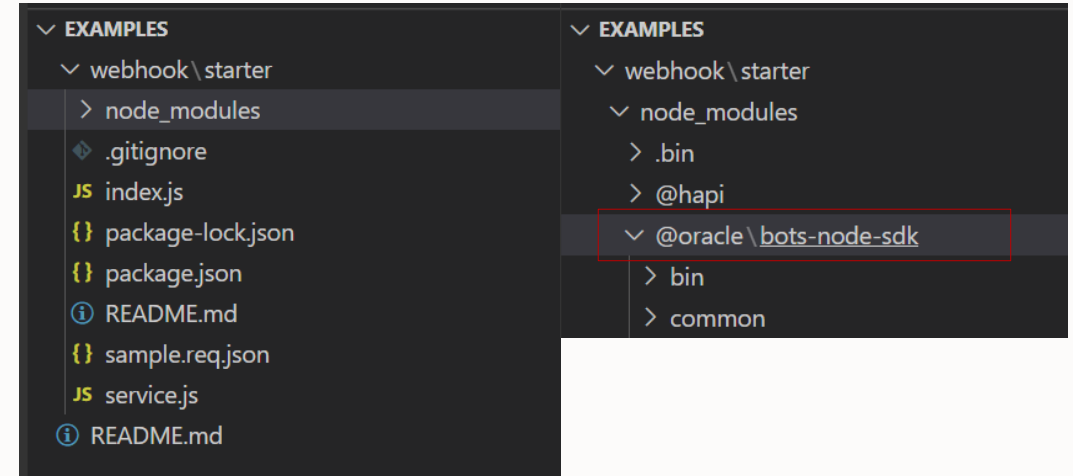
- Navigate to starter folder

Ensure Node and Node package manager are installed

- <https://nodejs.org/en/download/>

Issue `npm install` command

- Installs dependencies, e.g. Oracle Bots Node SDK



# Exploring the 'service.js' webhook client

```
19   webhook
20     .on(WebhookEvent.ERROR, err => logger.error('Error:', err.message))
21     .on(WebhookEvent.MESSAGE_SENT, message => logger.info('Message to bot:', message))
22     .on(WebhookEvent.MESSAGE_RECEIVED, message => {
23       // message was received from bot. forward to messaging client.
24       logger.info('Message from bot:', message);
25       // TODO: implement send to client...
26     });
27
28   // Create endpoint for bot webhook channel configuration (Outgoing URI)
29   // NOTE: webhook.receiver also supports using a callback as a replacement for WebhookEvent.MESSAGE_RECEIVED.
30   // - Useful in cases where custom validations, etc need to be performed.
31   app.post('/bot/message', webhook.receiver());
32
33   // Integrate with messaging client according to their specific SDKs, etc.
34   app.post('/test/message', (req, res) => {
35     const { user, text } = req.body;
36     // construct message to bot from the client message format
37     const MessageModel = webhook.MessageModel();
38     const message = {
39       userId: user,
40       messagePayload: MessageModel.textConversationMessage(text)
41     };
42     // send to bot webhook channel
43     webhook.send(message)
44       .then(() => res.send('ok'), e => res.status(400).end(e.message));
45   });
46 }
```

# Exploring the 'service.js' webhook client

```
19  webhook
20    .on(WebhookEvent.ERROR, err => logger.error('Error:', err.message))
21    .on(WebhookEvent.MESSAGE_SENT, message => logger.info('Message to bot:', message))
22    .on(WebhookEvent.MESSAGE_RECEIVED, message => {
23      // message was received from bot. forward to messaging client.
24      logger.info('Message from bot:', message);
25      // TODO: implement send to client...
26    });
27
28  // Create endpoint for bot webhook channel configuration (Outgoing URI)
29  // NOTE: webhook.receiver also supports using a callback as a replacement for WebhookEvent.MESSAGE_RECEIVED.
30  // - Useful in cases where custom validations, etc need to be performed
31  app.post('/bot/message', webhook.receiver());
```

## Webhook events:

- Error
- Message Sent
- Message Received
  - Under this event we need to implement the method to send the received messages from the bot towards the messenger client.



# Exploring the 'service.js' webhook client

- This block implements the POST route
  - This method is called to pass a user request – coming from the messenger client – to the bot
- The message needs to conform to the client message format.
  - *userId* and *messagePayload* defined by MessageModel

```
33 // Integrate with messaging client according to their specific SDKs, etc.
34 app.post('/test/message', (req, res) => {
35   const { user, text } = req.body;
36   // construct message to bot from the client message format
37   const MessageModel = webhook.MessageModel();
38   const message = {
39     userId: user,
40     messagePayload: MessageModel.textConversationMessage(text)
41   };
42   // send to bot webhook channel
43   webhook.send(message)
44     .then(() => res.send('ok'), e => res.status(400).end(e.message));
45 };
46 }
```

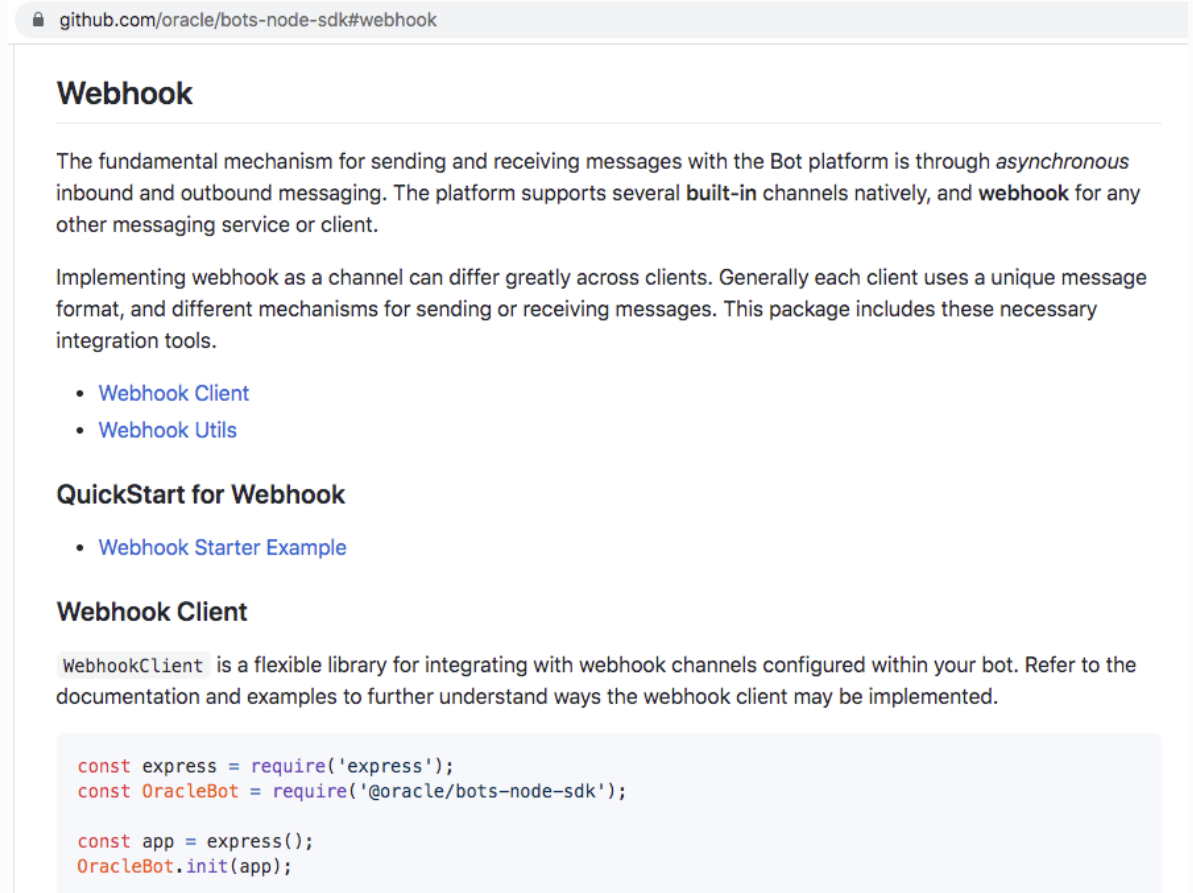
# Documentation

Available on Github

- <https://github.com/oracle/bots-node-sdk#webhook>

Oracle Digital Assistant documentation

- <https://docs.oracle.com/en/cloud/paas/digital-assistant/use-chatbot/webhooks.html#GUID-96CCA06D-0432-4F20-8CDD-E60161F46680>



github.com/oracle/bots-node-sdk#webhook

## Webhook

The fundamental mechanism for sending and receiving messages with the Bot platform is through *asynchronous* inbound and outbound messaging. The platform supports several **built-in** channels natively, and **webhook** for any other messaging service or client.

Implementing webhook as a channel can differ greatly across clients. Generally each client uses a unique message format, and different mechanisms for sending or receiving messages. This package includes these necessary integration tools.

- [Webhook Client](#)
- [Webhook Utils](#)

### QuickStart for Webhook

- [Webhook Starter Example](#)

### Webhook Client

`WebhookClient` is a flexible library for integrating with webhook channels configured within your bot. Refer to the documentation and examples to further understand ways the webhook client may be implemented.

```
const express = require('express');
const OracleBot = require('@oracle/bots-node-sdk');

const app = express();
OracleBot.init(app);
```

ORACLE