Business Chat
API Integration Tutorial

April 2020.1

Contents

Revision History

Version	Changes	Release Date
2020.1	Numerous clarifications of the human readable text based on feedback from technical inquiries and our issue tracking system.	April 2020
2019.6	Code was removed from tutorial. The code was rewritten in Python 3 and is in an attached zip file.	September 2019
2019.5	In Apple Pay exercises, language has been changed from `postalAddress` to `post` to correctly reflect the API.	June 2019
2019.4	Fixed broken links and other minor corrections.	May 2019
2019.3	Fixed broken links.	May 2019
2019.2	Minor code corrections.	April 2019
2019.1	Added Passing Authenticate Data exercise; updated Apple Pay exercises; removed Single Sign On; Incorporated FAQs and troubleshooting information into main document; minor content updates.	April 2019
2018.4	Incorporated corrections to the Apple Pay exercise; added Integrating the Sign Up Page and Sending Rich Links exercises.	September 2018
2018.3	Minor corrections to code examples. Updated troubleshooting section.	August 2018
2018.2	Added Created a Landing Page section and made minor content updates.	August 2018

Overview

Welcome to the Messaging Service Provider (MSP) API Integration Tutorial! At this point of the MSP onboarding process, you may either need to determine if your platform can support Business Chat functionality or you need to prepare your platform for a demo to the Apple Business Chat (ABC) team. Either way, the exercises in this tutorial can help you get your messaging platform ready by showing you how the functionality works.

For your demo, start with a brief overview of your messaging platform and its capabilities, provide the ABC team with a console view and, at the same time, an iOS device screen, so the ABC team can see both sides of the conversation. The ABC team also needs to see how you trigger objects, such as list pickers, time pickers, and the basic Apple Pay functionality.

You need to demonstrate that your messaging platform can perform each of the following fundamental business functionalities to qualify as a MSP.

✓	Feature	Reference
	Bi-directional messaging with text, images, and files	Receiving and Sending Messages
	List picker messaging, text-only and images	Sending List and Time Pickers
	Time picker messaging	Exercise: Sending a Time Picker
	Basic Apple Pay functionality	Apple Pay in Business Chat
	OAuth flow support	Authentication in Business Chat
	Custom iMessage app	iMessage Extensions
	Rich link messaging	Sending Rich Link Messages
	Agent/bot/user typing indicators	Typing Indicator Message
	"Close conversation" handling	Receiving Closed Conversation Messages
	Connect Landing Page	Connecting with Your Clients
	Group and intent ID-based routing	• Starting a Chat from your App • Adding a Business Chat Button to Your Website • Starting a Chat from a URL

Integrating with Business Chat

This is a step-by-step tutorial, designed to help you integrate your Customer Service Platform (MSP) with the Business Chat API. The Business Chat Team structured the tutorial so you could follow the API in the documentation on Apple Developer.

NOTE For the purposes of this tutorial, configurations and implementations of all Business Chat features will occur on your test server.

Although the provided code examples are in the Python programming language and HTTP protocol is the fundamental architecture, you can adapt the code for use in any language.

The Business Chat Sandbox is a platform that allows customers to send and receive messages using Business Chat. The Sandbox is a server with a MSP and the Business Chat Service, you do not need to connect your platform to it for it to work.

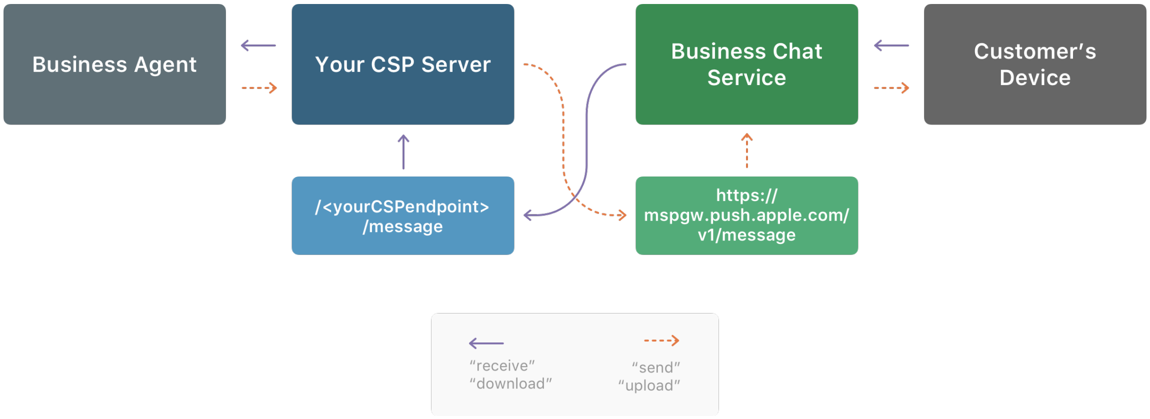
This tutorial relies on a server provided by you and configured as a Customer Service Platform provider. This is a test server, not a production server. Your MSP test server needs to access to the web so it can talk to the Business Chat Service.

You will learn the following in this tutorial:

- Experiment with different message types, from basic text messaging to sending Apple Pay payments.
- Explore using different message types, such as text and interactive, and the handling of small to large attachments.

By the end of this tutorial, you should be familiar with the concepts and features Business Chat offers to businesses, including familiarity with implementing these features on your MSP.

All tutorials use the following diagram when sending and receiving data. A customer's device sends a message to the Business Chat service which then posts the customer's message to the MSP endpoint. The MSP digests the message, forwarding it to the appropriate business agent for a response. The Business Agent responds sending the message back to the MSP who then posts it to the Business Chat Service and from there it is forwarded back to the Customer's Device.



Setting Up Your Accounts on Apple Business Register

Your MSP, Business Chat accounts, brands, team members, and organization details must all be entered into Apple Business Register (ABR) to create the accounts needed for this tutorial. The link for ABR is at:

https://register.apple.com/

ABR is designed as a self-service portal. It is mostly self-explanatory with a few key items of note:

- You will have to apply for, and be approved for, your company to be able to offer MSP service. If you do not see Customer Service Platform as an item in ABR, contact your Business Chat account manager to determine the application process
- ABR will generate several key values that you will need: CSP ID, Business ID, and secret passphrase. Other parameters, you will have to enter into ABR, such as organizational details, team member details, brand colors, brand logos, and URLs to your server(s)
- Each Business ID is associated with one and only one CSP ID. Internal test Business IDs have no brand. Commercial Business IDs must be associated with a brand in ABR, which will require an approval step by the brand review team that may take several business days

Configuring Your MSP Test Server

When prompted, you can refer to the following information throughout the exercises in this tutorial:

- CSP ID***:** a unique UUID received when your MSP account has been approved
- Business ID*: a unique UUID received when your Business Chat account has been approved
- Secret passphrase for API*: a Base64-encoded string
- MSP message endpoint: `https://<your.endpoint.host>/message`

* Information obtained from your Apple Business Register MSP account.

NOTE Messaging Service Provider (MSP) were previously referred to as Customer Service Provider (CSP). The code samples refer to CSP ID which may be referred to MSP ID in other documents.

MSP endpoint SSL certificate requirements

Your SSL certificate must meet the HTTPS requirements needed to connect to the Business Chat endpoints. Your SSL certificate should contain the following:

- Valid SSL certificate
- Issued name on the SSL certificate must match the hostname
- Certificate must include the entire trust chain, including CAs and root
- Certificate must be in Oracle Java 8+ framework's default trust store, meaning signed by trusted CAs, supported encryption, and format.

NOTE ngrok hosts, and all similar firewall-defeating technologies, are not allowed for use with Business Chat. They will be declined or disabled for messaging, client landing page, or authentication endpoints.

Business Chat server endpoints

The endpoints allow messages to be routed between the Business Chat service and your platform. See How to Setup Your Endpoint.

- Production: `https://mspgw.push.apple.com/v1/message`
- Staging: `https://mspgw-int.push.apple.com/v1/message`
  The staging endpoint is for the development and testing of server changes. Even though you can message the actual users through the staging endpoint, it has a load limit and is not intended for production messages.

TIP Business Chat requires that the URL for your messaging requests end in `/message`.

Tools

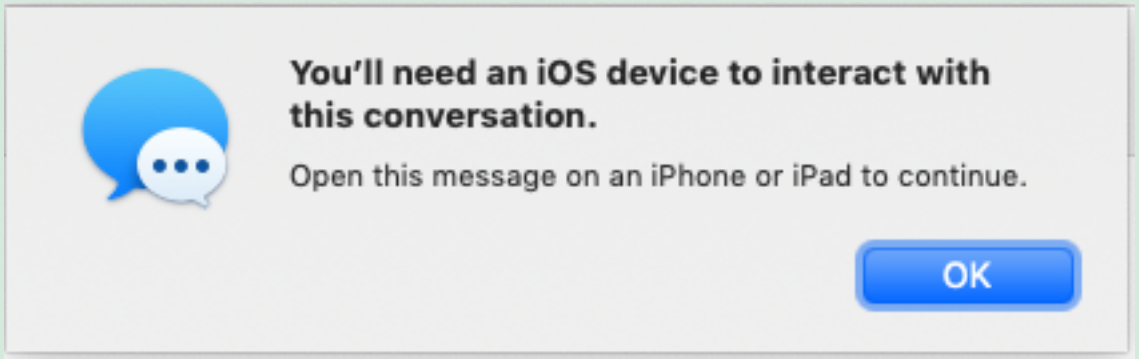
This section includes the tools used throughout this tutorial.

Apple iOS devices

To test and verify the exercises from both your console and customer-side, you'll need an Apple supported device that can run iOS 11.3 or later.

TIP If your device continues to display Waiting for activation, use the following link to check your device settings at https://support.apple.com/en-us/HT201422.

NOTE Business Chat is available from macOS devices with a limited feature set. Currently text messages, attachments, authentication, and Apple Pay payments are available. Other message types will appear in the Messages window, but display the following message when clicked:



You'll need an iOS device to interact with this conversation.

Open this message on an iPhone or iPad to continue.

Development tools

The following development tools are needed to complete the exercises in this tutorial:

- Python 3.7
- Python modules:
  - Flask

- pyjwt
- pycrypto
- requests
- cryptography

How to Setup Your Endpoint

If your endpoint isn't set up, you'll need to set up a messaging server platform endpoint to send and receive messages from the Business Chat service.

1. Select a server that has sufficient storage available for the application, as Apple does not save messages after delivery.
2. Set up your server to receive HTTPS messages.
3. Configure a URL endpoint on your messaging server. Your server must be capable of receiving traffic on `/message`.
4. Confirm that you can receive traffic to your URL on `/message`.
5. You must have a logging tool installed and activated so that you can triage issues from your server side.

Setup an HTTPS proxy

The Business Chat Server expects the MSP endpoint to listen for the registered platform's API base-URL supporting HTTPS. You can simplify local development by setting up a proxy on the platform's API base-URL server to point to a local HTTP port, such as 8002:

- `https://your.endpoint.base.url` points to http://localhost:8002

NOTE Exercises in this tutorial expect that you are redirecting messages to the Business Chat Server at:

`http://localhost:8002`

You can send messages to localhost, but you cannot receive them.

Alternatively, you can set up *SSL support for Flask* (search the web for tutorials and resources).

```
from OpenSSL import SSL
context = SSL.Context(SSL.SSLv23_METHOD)
context.use_privatekey_file('your_server.key')
context.use_certificate_file('your_server.crt')

app.run(host='127.0.0.1',port='12344', debug = False/True,
ssl_context=context)
```

Receiving and Sending Messages

When a customer sends a message to a business, the customer's device sends the message to the Business Chat server. The Business Chat server forwards the message to the Messaging Service Provider (MSP) by making a `POST` request to the `/message` endpoint. For more information on receiving messages, see Messages Received.

In this section, you'll learn how to listen, receive, and validate incoming messages.

TIP You can initiate a conversation in the Messages app by using the business token directly as a recipient in Messages. The format is `urn:biz:<your-business-id>`.

When a business replies to a customer's text message, the Messaging Service Platform (MSP) forwards the reply to the Business Chat server. To forward the reply, the MSP prepares the message for delivery, and sends a `POST` request to the `/message` endpoint hosted by Business Chat. For more information on sending messages, see Messages Sent.

Exercise: Listening for Incoming Messages

In this exercise, you run a simple web service and listen to message requests. Send a text message from the Messages app on a device to your business and observe the request of the text message.

TIP Listening for incoming messages is the only way to get the opaque ID. An opaque ID is an identifier used for anonymous routing between the customer and the business.

On your server

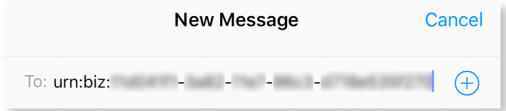
1. Locate the zip folder and unzip it.
2. Locate and run the *00_listening.py* file.
3. Listen to `POST` requests at: `https://<yourCSPEndpoint>/message`

Sending a message from your iOS device

Send a text message from the Messages app on an iOS device to your business.

TIP When a customer ends the conversation by deleting the message, a conversation type `close` message is sent to your MSP. You cannot begin a new conversation with the customer.. Only the customer can re-initiate the conversation. When they do, however, they will return with the same opaque user ID as before.

1. From an iOS device, open Messages.
2. Put in your business URN identifier (`urn:biz:<your registered business id>`) as recipient.


New Message

Cancel

To: urn:biz:[redacted]

3. Compose a simple text message, such as “Hello business!”, and tap send.

Expected MSP server response

By default, on your MSP Test Server you’ll receive at least two messages: one is for typing start, and the other is for the actual message. The typing indicator in the output is automatically displayed on the console and is not a part of the `typing_start` indicator. Observe the request of the text message.

```
Just received a message!
Just received a message!
```

Exercise: Receiving a Text Message

In this exercise, you receive, decode, and read an incoming text message.

TIP Once a message has been delivered, you cannot retrieve it.

Complete the following tasks:

1. Send a text message from the Messages app to your business.
2. Receive the message payload.
3. Uncompress, using GZIP, and print the payload.

On your server

1. Locate and run the *01_receiving_text_message.py* file.
2. Once the file has completed running, save the `source_id` for use later in the tutorial.

Expected MSP server response

The type of information given is the *opaque user ID* and the *device agent*. The opaque user ID is the source ID when receiving messages. The device agent tells you the type of device the customer is using, such as iPhone OS, Mac OS X, Watch OS, and Unknown.

```
User is typing...
Just received a text message!
Message body: Nice to meet you
Source ID: <opaque user ID>
Device Agent: <device agent of user>
```

NOTE Write down the `Source ID` as displayed in the output above. You'll use this as the `destination_ID` in future exercises when sending messages.

TIP The opaque user ID used in this section if very important to the functionality of Business Chat. It persists over long periods of time (years). It can be used across multiple devices by the user. It is unique identifier used between one Business ID and one user. It is not shared between businesses, even on the same MSP. If the user initiates a conversation with another Business ID, she will receive a new opaque user ID for that conversation.

Exercise: Validating a Received Message

In this exercise, you verify the authorization header of a received message. For more information on validating messages, see Validating a Message.

Complete the following tasks:
1. Extract the JSON Web Token (JWT) string from headers.
2. Verify the signature and audience by decoding the JWT string.

When receiving a message on your messaging platform, the JWT audience (`aud`) value in the JWT payload, is your **CSP ID**. Do not use your Business ID to validate.

TIP HTTP header field names are case-insensitive, according to https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html.

On your server

1. Locate and run the *config.py* file to completion.
2. Locate and run the *verify_message.py* file to completion.

Expected MSP server response

By default, you receive at least two messages: one is for typing start, and the other is for the actual message. The typing indicator in the output is automatically displayed on the console and is not a part of the `typing_start` indicator. Observe the request of the text message.

```
Authorization succeeded.
Authorization succeeded.
```

If you are not receiving any requests from Apple Business Chat server, check the following:
- Make sure your endpoint SSL certificate fulfills the requirements. See Configuring Your MSP Test Server.
- Make sure your server is listening to `POST` requests on `https://<your.domain>/<path>/message`
- Make sure you are using the iMessage service for outgoing messages. You can check the iMessage setting by going to Settings > Messages.

- Make sure your messages actually got sent from the device. If you are seeing a progress bar that runs for a long time, please check your internet connection and iMessage settings at https://support.apple.com/en-us/HT201422.
- Make sure there is no software blocking requests or closing the connection. Be aware that your service may be blocking incoming requests due to strict rules. One common case is, in the "hostname" HTTP header, the port number is included, even though it's optional.

Exercise: Sending a Text Message

In this exercise, you send a text message payload.

Complete the following tasks:

1. Successfully derive the binary CSP secret from secret passphrase.
2. Create a valid JWT header using your CSP secret. See Authorizing Messages.
3. Get the `destination_id` from the response in Exercise: Receiving a Text Message to respond.
4. Copy and paste the `destination_id` into the Listing send_text_message.py code.
5. Assemble a message header and payload.
6. Send a message.

When sending a message from your messaging platform, the JWT issuer value (`iss` in the JWT payload) should be the *CSP ID*. Using Business ID in this instance results in authorization failures.

TIP The timestamp unit, `iat` in the JWT payload, should be in *seconds*, not milliseconds.

The following JWT header creation code makes the authorization header field token.

On your server

1. Locate and run the *jwt_util.py* file to completion.
2. Locate and edit the *03_send_text_message.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
3. Save and run the *03_send_text_message.py* file.

Expected MSP server response

You should see the following message on your MSP server.

```
Business Chat server return code: 200
```

If you are unable to send a text message, then check the following:

- Go through the Business Chat documentation and the tutorial.
- Examine the response body for potential errors.
- If you get an error code, refer to the table below.

Error	Troubleshooting Tips
401 "unauthorized" error	• Make sure you are using your **CSP ID** to encode the JWT token, and your **Business ID** in headers and payload. • Make sure to Base64-decode the textual **secret** when generating your JWT token. • Ensure your JWT timestamp (`iat` value) is in **seconds** not in milliseconds or another unit. • Check you have the "Bearer" prefix in authorization header.
410 "session expired" error	The customer you are attempting to message deleted the conversation from the messages tray on their device.

Use the following matrix to troubleshoot any whitelisting problems you may encounter.

Apple ID	White-listed	Closed Conv.	Device to MSP	MSP to Device
✓	✓		Ok	Ok
✓			Not delivered	Not delivered
✓	Removed		Ok	404 Resource Not Found
Logout*	✓		Ok	404 Resource Not Found
			n/a	No email sent
		✓	Ok, reopens conversation.	410 Resource Gone

* Occurs only if the customer is logged out on all devices for a particular Apple ID. Otherwise, messages are delivered to devices still logged in with the Apple ID.

Exercise: Sending an Image Attachment

A message can include one or more attachments. An attachment can be an image, PDF, or other file type, and it must be smaller than 100 MB. For more information about attachments, see Sending Messages with Attachments.

In this section's exercises, you download different types of attachments. In this exercise, you send a message with an image attachment.

1. Perform a pre-upload request to retrieve an attachment upload URL.
2. Encrypt the attachment data.
3. Upload the encrypted attachment data to the retrieved attachment upload URL.
4. Send the message with metadata of the attachment.

On your server

Locate and run the *attachment_cipher.py* file.

Send the message with Base64-encoded images

In this exercise, you send a message with Base64-encoded image attachments. In the message payload body, you need an Unicode Object Replacement Character (`\uFFFC`) as a placeholder for each attachment you send.

On your server

1. Locate and edit the *04_send_image_attachment.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
2. Save and run the *04_send_image_attachment.py* file.

Expected MSP server response

You should see the following message appear on your MSP server.

```
Business Chat server return code: 200
```

Expected client device response

If the encrypted data does not display correctly in the Message client, check your encryption implementation.

If attachments are not showing up in the Messages app, check the following:

- Make sure in the message body you have a *Unicode Object Replacement Character* (`\uFFFc`) for each attachment you are sending
- Make sure you are using Base64-encoding for the attachment signature
- If the sent images are displayed as icons in the Messages app, then verify the following:
  - Encrypt the attachment using the correct algorithm
  - Do not compressed the data when sending
  - Uploaded data is not corrupted
  - Do not subsequently modify the data using network nodes

To verify that your encryption implementation is correct:

In your encryption method use the following encryption key rather than a randomly generated one: `12E9F08B6B0CCC36DF688BF167FAF7BF3E7E696D1A66E98C9E9949131C9F8E07` (hex-encoded).

1. Hex-decode the string.
2. Encrypt the string "12345" using the above listed encryption key.
3. The Base16-encoded result should be `7634125F65`.

To verify that your decryption implementation is correct:

Encrypt then decrypt any given string. The result should be the same as the input.

Exercise: Downloading Attachments

In this exercise, you receive a message containing attachments. You need to download and decrypt the attachments. See Downloading and Decrypting an Attachment.

TIP In general, the URL field in the attachments array should not be URI-encoded. You can include a mix of unescaped and escaped characters if that is how the client device uses the link.

Complete the following tasks:

1. Receive a message payload with attachments.
2. For each attachment perform a `/preDownload` request to get the download URL.
3. Convert the provided hexadecimal signature from the payload to a Base64-encoded signature.
4. Download the attachment using the download URL.
5. Decrypt attachment data and save to file.

Save the message attachments as local files under the current path.

NOTE You always get a download URL from the `/preDownload` step even if the parameters are not correctly set. However, the downloaded data is not valid in such a case.

On your server

Locate and run the *05_downloading_attachments.py* file.

Expected MSP server response

```
2 attachments found in the message.
writing to local file: 52106351067__A31A08AE-A449-4EDD-A735-458D17ADF9EA.JPG
writing to local file: 52106351346__8CEE7676-0E8C-4D19-83B5-C680836837CC.JPG
```

Error codes

Error	Troubleshooting Tip
400 “not authorized”	Make sure you are using the correct encoding for attachment signature. It should be a Base64-encoded string.
400 error when calling `/decodePayload` for interactive data payload	Make sure you have “bid” as a header in your request.
Problems successfully decrypting the attachment	Make sure you are using the correct algorithm to decrypt. Once you are confident that the encryption algorithm works as expected, you may try to encrypt a file, and immediately decrypt the data. If the file can be recovered, then the decryption works as expected.

Ensuring Signature Fields are Correct

There are several signature fields used by Business Chat, some of which are base 16 and others are base 64 encoded fields.

TIP

1. When receiving a message from Business Chat, the signature field of the attachment is hexadecimal (base 16) encoded
2. When sending the signature header in the preDownload request, the value is base64 encoded
3. When sending a message from your MSP to Business chat, the "signature" field is actually called signature-base64 and is base64 encoded

Sending List and Time Pickers

Interactive messages provide a range of functionality, from letting the customer select an item from a list to scheduling an appointment. Business Chat comes with a set of interactive messages that you can use, or you can create and use custom interactive messages. When sending an interactive message to the customer, set the `type` field to `interactive`. For more information, see List Picker and Time Picker.

In this section's exercises, you learn how to provide a rich user experience using list and time pickers.

Exercise: Sending a Text-only List Picker

In this exercise, you generate a request ID to send a text-only list picker payload to trigger an item selection in the Messages app.

The request ID ensures that the requests and responses are properly paired. In situations where an agent sends multiple interactive messages, the MSP may receive responses out-of-order or no response for some requests. The request ID ensures that the responses match the corresponding request.

TIP When parsing the list picker response, note that the selected options of the list picker are sent back to the MSP as a subset of the options in the list picker.

Complete the following tasks:

1. Generate a request ID.
2. Assemble list picker item configuration with optional sections.
3. Send a message with payload.

On your server

1. Locate the code zip folder and unzip it.
2. Open and edit the *06_send_text_list_picker.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
3. Save and run the *06_send_text_list_picker.py* file.

Which version of version?

The 'v': 1 key-value pair and the 'version': '1.0' pair both indicate version numbers. The 'v' indicates the version of the Business Chat API, corresponding to the endpoint version at https://mspgw.push.apple.com/v1. The 'version' inside of the interactive message indicates the version of the list picker implementation that is requested.

Currently both have only a single version. We architected the system, however, to up-rev the list picker and other interactive message components as a higher frequency than the underlying API.

Also note that the 'v' version is an integer data type, while the 'version' is a string.

Expected MSP server response

```
Business Chat server return code: 200
```

If your custom message isn't being shown on your client device, use the following matrix identifying the type of messages sent and how they appear in the UX.

Message Type	**Reply Message**		**Response Message**	
	Text	**Picture**	**Text**	**Picture**
List Picker	✓	✓	✓	✓

Exercise: Sending a List Picker with an Image

In this exercise, you send a list picker with an image payload to trigger an item selection in the Messages app. When customer's responds to a list picker with a text message, the two messages are delivered separately without association.

> **TIP** If you cannot see the images in interactive message bubbles, make sure your image DPI is set to 72, which is the common DPI setting for iOS.

Complete the following tasks:

1. Base64-encode the image data.
2. Assemble a list picker payload with the encoded image data.
3. Send a list picker message with the payload.

On your server

1. Locate the code zip folder and unzip it.
2. Locate and edit the *07_send_list_picker_with_image.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
3. Save and run the *07_send_list_picker_with_image.py* file.

Expected MSP server response

```
Business Chat server return code: 200
```

Exercise: Sending a List Picker with Multiple Images

In this exercise, you send a list picker with multiple images and receive a data reference for use in the next exercise, Exercise: Sending a List Picker Using a Data Reference.

Complete the following tasks:

1. Perform iteration over a dictionary containing three image files, encrypting each for transmission.
2. Use references in JSON to specify which images go with each section of the list picker.
3. Print out the data reference in JSON format for use in the next exercise, Exercise: Sending a List Picker Using a Data Reference.

Use the same List Picker from Exercise: Sending a List Picker with an Image. Note the references specify multiple images. You use a dictionary to match the image identifiers in JSON to the file names. Then, loop through the images to encrypt each one and add it to the payload.

This example includes an additional header parameter, `include-data-ref`, that is set to `true`. This setting directs the Business Chat server to return the data reference. Because HTTPS header key-value pairs are string only, set the value, `true`, to a four character string.

On your server

1. Locate and edit the *08_send_list_picker_with_multiple_images.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
2. Save and run the *08_send_list_picker_with_multiple_images.py* file.
3. Save the data reference from the output to use in later exercises.

Expected MSP server response

```
Business Chat server return code: 200
Business Chat server response body:
{
    "dataRef": {
        "bid":"<bid save for next exercise>",
        "owner":"<owner save for next exercise>",
        "url":"<url save for next exercise>",
        "size":"<size save for next exercise>",
        "signature":"<signature save for next exercise>",
        "signature-base64":"<signature-Base64 save for next exercise>",
        "title":"Select Produce",
        "dataRefSig":"<dataRefSig save for next exercise>"
    }
}
```

Exercise: Sending a List Picker Using an Interactive Data Reference (IDR)

In this exercise, you reuse a previous interactive data reference to send a list picker.

An Interactive Data Reference in Business Chat is like a super-charged URL. It’s a pointer to a set of large (usually greater than 10KB, but can be 100MB or more) media assets that is easier to use rather than expending the bandwidth to upload or download assets. It conforms to the JSON standard in a form that is unique to Business Chat.

The IDR allows you to send the list picker using an abbreviated handle instead of uploading all the attachments again.

On your server

This exercise is based on the list picker sent in Exercise: Sending a List Picker with Multiple Images, and that you got the data reference from the response. Make sure the `dataRefSig` field is present in the data reference.

1. Locate and edit the *09_send_list_picker_with_data_ref.py* file with the following values:
    a) Insert the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
    b) Insert the `interactive_data_ref`, located towards the bottom of the file, with the output data reference from Exercise: Sending a List Picker with Multiple Images.
2. Save and run the *09_send_list_picker_with_data_ref.py* file.

Expected MSP server response

```
Business Chat server return code: 200
```

Exercise: Sending a Time Picker

In this exercise, you send a message payload to trigger a time picker in the Messages app on the customer's device.

When setting the times for your list picker, the start date cannot be earlier than the current date/time, otherwise the times are removed from the time picker. If all items in the time picker are for past times, then the device displays an empty time picker with a message "No times available." You can have the following types:

- long duration items, even extending to days long
- zero time length duration items, use this feature with discretion
- across different days, different months or different years

Complete the following tasks:

1. Generate time picker request ID.
2. Assemble time picker configuration with or without timezone offset.
3. Send message with payload.

Sending a time picker using the customer's timezone

In this example, you are sending the time of an event happening in San Francisco on 2020-10-15 at 10:00am (PDT: GMT-7 hours), and you want the customer's Messages app to display the time according to the customer's time zone setting. For example, for a customer in New York (EDT: GMT-4 hours) the time picker would display 2020-10-15 at 1:00pm.

Sending the time of an event requires you to convert the event time to GMT, `2020-10-15T17:00+0000`, and use it as the `startTime` value. In this case, you do not set the `timezoneOffset` value.

On your server

1. Locate and edit the *10_send_time_picker_with_user_timezone.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
2. Save and run the *10_send_time_picker_with_user_timezone.py* file.

Sending a time picker using a fixed timezone

In this example, you are also sending the time of an event happening in San Francisco on 2020-10-15 at 10:00am. However, you want the customer's Messages app to display 2020-10-15 at 10:00am regardless of the customer's timezone.

Setting the time of the event regardless of the customer's timezone requires you set the `startTime` value as GMT, `2020-10-15T17:00+0000`. And you need to set a `timezoneOffset`, in this case `-420` minutes for San Francisco PDT.

With the `timezoneOffset` set, the customer's device always displays the time in that specific timezone without converting the time to the customer's timezone.

On your server

1. Locate and edit the *11_send_time_picker_with_fixed_timezone.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
2. Save and run the *11_send_time_picker_with_fixed_timezone.py* file.

Expected MSP server response

```
Business Chat server return code: 200
```

If your custom message doesn't show on your client device, use the following matrix identifying the type of messages sent and how they appear in the UX.

Message Type	Reply Message		Response Message	
	Text	Picture	Text	Picture
Time Picker	✓	✓	✓	✓

Advanced Interactive Messaging

You can use an iMessage extension to give your customers an interactive experience that is unique to your business.

Exercise: Retrieving a Large Message Using an Interactive Data Reference

In this exercise, you learn how to decrypt, decode, and parse a data reference that results from the customer response to an interactive message.

NOTE This exercise is the most difficult for most engineers who implement Business Chat. Once you have this one working, it will get easier from there.

A List Picker response includes all of the images of the original picker. This ensures that the response JSON has a complete set of digital assets to render the bubble, images, user selection, and future interaction. The other interactive messages work this same way.

When the response payload exceeds 10kB, Business Chat saves the full JSON to a cloud server and the MSP receives an Interactive Data Reference (IDR). The IDR has URLs to retrieve the full response payload and decryption keys to decipher the response. The response that your MSP receives is an Interactive Data Reference (implied by the "interactiveDataRef" key):

https://developer.apple.com/documentation/businesschat/enhancing_the_customer_s_user_experience/receiving_large_interactive_data_payloads

Your messaging platform must parse the IDR and use it to download the full JSON of the response. Use the full JSON to complete the action of interactive message.

On your server

1. Locate the code zip folder and unzip it.
2. Locate and run the *12_large_interactive_message.py* file to set up the listener.
3. Go through the Exercise: Sending a List Picker with Multiple Images to send a list picker to your test device.
4. On your test device, make your selections, enter a text message and send it.
5. Observe the interaction between your MSP and the Business Chat service.
6. Open the file `data_reference_debug.json` in your favorite text editor to see the full payload retrieved by script.

If your custom message doesn't show on your client device, use the following matrix identifying the type of messages sent and how they appear in the UX.

Message Type	Reply Message		Response Message	
	Text	Picture	Text	Picture
Custom Interactive Messages	n/a	n/a	*	*

* Encoded as an attachment rather than interactive message data

n/a = Not applicable as there is no generated response message

Apple Pay in Business Chat

When a business asks for payment from a customer who is purchasing goods and services through Business Chat, the customer can use Apple Pay to make the payment. For more information, see Apple Pay in Business Chat.

To support refunds to Apple Pay purchases, use a workflow outside of Business Chat. For more information, see About Apple Pay for merchants.

Exercise: Send an Apple Pay Request—Basic Functionality

The following Apple Pay exercise covers the basics needed for you to send an Apple Pay request and how to read the return message. To qualify as a MSP, you'll need to demonstrate the basic Apple Pay functionality on your platform. **PRO TIP** When you host the merchant payment token generator (called payment gateway here), you will receive a payment transaction token. For the sake of running a demo, you can simply retain the token and ensure that no payment transaction is executed. This will allow you to run tests and demo your implementation without incurring charges through your Apple Pay account.

Complete the following tasks:

1. Setup and configure a merchant account with ID and certificates.
2. Enter your Merchant information into Apple Business Register to link your Apple Pay Merchant account with your Business Chat service.
3. Run a test payment gateway that does not actually process any payment.
4. Start a test payment session from the Apple Pay gateway.
5. Send a test payment request as a message using Business Chat.
6. Accept a test payment on iPhone.
7. Receive the test payment at the gateway.
8. Receive the Business Chat interactive message indicating success or failure of the payment request.

For more information on sending Apple Pay requests, see Sending an Apple Pay Payment Request.

Step 1: Set up a merchant account

If you already have an Apple Pay Merchant account, skip ahead to step 4 and download the Merchant items.

1. You must have an Apple Developer account in order to create an Apple Pay Merchant account. If you do not have an Apple Developer account, please set one up at developer.apple.com. The approval time may take several days.
2. Use one of the following tutorials to add Apple Pay Merchant capabilities to your Apple Developer account:
    - Apple Pay "Configuring Your Developer Account" tutorial video:
        - https://developer.apple.com/videos/play/tutorials/configuring-your-developer-account-for-apple-pay/
    - "Configuring Your Environment" section in Apple Pay JS Documentation:
        - https://developer.apple.com/documentation/apple_pay_on_the_web/configuring_your_environment
3. Create a Merchant ID, a Payment Processing Certificate, and a Merchant Identity Certificate in your Apple Developer account.

4. Make a note of your Merchant ID from your account.
5. Download the Payment Processing Certificate and Merchant Identity Certificate.

Step 2: Configure your Apple Business Register account

Use the following steps to enter your Merchant Information into your Business Chat account:

1. Go to register.apple.com, and sign in with your Apple ID as the administrator or technical contact for the business that owns your Apple Pay credentials.
2. Go to your company's Business Chat Accounts and find the appropriate business account. If your company has multiple accounts (as most do), ensure that you enter the Merchant information into the Business Chat account corresponding to your company's Apple Pay capability.

NOTE Each Business Chat ID can have only one Apple Pay Merchant ID.

3. Find the Apple Pay section in account configuration (as shown below).
4. Click on Edit.
5. Enter the Merchant ID into the appropriate field, and submit for review.
6. Once the updated configuration is active, you are able to use the Merchant ID in Business Chat.

Step 3: Register your merchant ID

Use the following steps to verify that your Merchant ID is registered at Apple Business Register website:

1. Go to register.apple.com, and sign in with your Apple ID as the administrator or technical contact for the business that owns your Apple Pay credentials.
2. Go to your company's Business Chat Accounts and find the appropriate business account. If your company has multiple accounts, ensure that you enter the merchant information into the Business Chat account corresponding to your company's Apple Pay capability.
3. Find the Apple Pay section in account configuration (as shown below).
4. Click on Edit.
5. Enter the Merchant ID into the appropriate field, and submit for review.
6. Once the updated configuration is active, you can use the Merchant ID in Business Chat.

Step 4: Generate the PEM file required by the payment session endpoint

This step is necessary to work with the sample code provided in this document. If you have an Apple Pay endpoint already in production and are using it in this exercise in that environment, then you may not need to follow this procedure.

We use command line tools for this procedure. If you are on macOS and familiar with Keychain Access, you can perform these steps using that application instead.

1. Create a directory for your key store.

```
mkdir keysncerts
cd keysncerts/
```

2. Initiate keystore with RSA 2048 key pair and generate a signing request.

```
keytool -genkeypair -keystore fileforkeys.p12 -storetype pkcs12 -alias
keyone -keyalg RSA -keysize 2048
```

```
keytool -certreq -keystore fileforkeys.p12 -storetype pkcs12 -alias keyone
-sigalg SHA256withRSA > my_request.csr
```

3. Generate an Apple Pay Merchant Identity certificate.

   Log into your Apple Developer account on developer.apple.com. Navigate to your certificates. Then upload my_request.csr to developer portal, generate the Merchant Identifier certificate, and it will download as merchant_id.cer.

```
openssl pkcs12 -in fileforkeys.p12 -out pv.key -nodes -clcerts
```

   Move merchant_id.cer over to the keysncerts directory. Then convert and export it as a PEM file.

```
openssl x509 -inform DER -outform PEM -in merchant_id.cer -out
server.crt.pem
```

4. Generate the payload file.

```
vi data.json
```

   Enter the file, save, and exit.

```
{
    "merchantIdentifier": <SHA256 hash of your text merchant identifier>,
    "domainName": <domain name associated with Apple Pay merchant account>,
    "displayName": <Merchant name in human readable form>,
    "initiative": "messaging",
    "initiativeContext": "https://<...your endpoint URL...>/paymentGateway"
}
```

5. Issue the command to get your Merchant Payment token.

```
curl -k -vvvv --request POST -d "@data.json" --header "Content-Type:
application/json" --cert server.crt.pem --key pv.key "https://apple-pay-
gateway.apple.com/paymentservices/paymentSession"
```

If you are able to retrieve the token from the command line above, use the pen file and private key generated here on your server.

Step 5a: Run a test payment gateway

Run a test payment gateway to verify that you can receive payment requests from the customer. For more information about the payment gateway API, see PKPaymentAuthorizationController.

On your server

1. Locate the code zip folder and unzip it.
2. Locate and run the *13_test_payment_gateway.py* file.

Expected MSP server response

```
Payment received!
Request Identifier: <request identifier in message payload>
Payment Method Dictionary: {'displayName': 'Visa 1234', 'type': 'Credit',
'network': 'Visa'}
```

Step 5b: Send Apple Pay Request

Respond with a test payment session to a request from the test payment gateway, using the following fields in the payload:

- `merchantIdentifier` SHA256 hash represented in hexadecimal of merchant identifier
- `displayName` Your merchant name in text (human readable)
- `domainName` Your base website without any punctuation or prefix (example: "developer.apple.com"). This must match the `domainName` for your merchant account.
- `initiative` Set to `messaging` for Business Chat
- `initiativeContext` Your payment gateway URL, with the `https://` prefix. This is a publicly exposed network endpoint. It does not work from a private network.

The `domainName` field is used for authentication versus the merchant information in Apple Pay, so the HTTPS prefix and punctuation is not used. The `initiativeContext` field defines an endpoint called during the transaction, so the HTTPS prefix and punctuation are required.

On your server

1. Locate and edit the *14_send_apple_pay_request.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
2. Save and run the *14_send_apple_pay_request.py* file.

Step 5c: Enhance your Apple Pay Request

In the previous exercise you successfully sent an Apple Pay request. The appearance, though, was very plain. In this exercise, you'll learn a few techniques to enhance your Apple Pay transactions on Business Chat and use the full features of this product.

When reading the code, you should notice that it can recognize and parse the payment response.

The next change is the new handlers for new endpoints. The use cases show you how businesses can update the shopping cart depending on a customer's payment method, shipping method, or contact information. For illustrative purposes, we made all of these endpoints on our main domain. They can be URL endpoints to different domains for different functions.

Below are the different endpoints:

Endpoint	Description
`/fallback`	Captures the response for devices that are not able to use Apple Pay. The request redirects the client device to this URL which allows you to explain a procedure for completing the transaction through an alternative payment medium.
`/orderTracking`	Captures changes to the line items, amounts, or status for each line item. If set, it calls on the first presentation of the line items to the user.
`/paymentMethodUpdate`	Called when the payment method, such as debit, credit, or cash, changes. For more information, see ApplePayPaymentMethodUpdate.
`/shippingContactUpdate`	Called when any of the shipping contact details, such as shipping address, associated email address, or associated phone number, change. Also called when the customer taps their device to initiate an Apple Pay session. For more information, see ApplePayShippingContactUpdate.
`/shippingMethodUpdate`	Called only when the shipping method is changed. For more information, see ApplePayShippingMethodUpdate.

Step 5d: Add images to your Apple Pay Request

By adding a Base64-encoded image of their order, customers can visually verify their purchase. For more information about image encoding, see Exercise: Sending a List Picker with an Image.

In this exercise, two fields have been added that require the customer to provide additional information to complete the transaction. The fields, `requiredBillingContactFields` and `requiredShippingContactFields`, are arrays that require the customer to provide a postal address, name, and phone. We recommend adding these fields for physical goods. For virtual goods, fewer fields may be more appropriate. For more information about the list of available values for billing and shipping contact fields, see ApplePayContactField.

TIP On the client device, these required fields default to the information defined in Settings > Wallet & Apple Pay. The user can overwrite these fields, though, for any particular transaction.

To make up the payment request, multiple line items have been added. These items appear in the order given in the payload on the client device in the Apple Pay drawer for Business Chat.

Set customer expectations about discounts or charges by using a line item for each charge or discount. Each line item can be a zero, positive, or negative value to indicate the type of charge or discount. The total amount of the order must be greater than zero.

The amounts are specified by strings. This allows flexibility across different currencies and payment systems. For more information about payment line items, see lineItems.

On your server

1. Locate and run the *15_handle_payment_gateway.py* in the background to set up the server with the endpoint listeners.
2. Locate and edit the *16_send_rich_apple_pay_request.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
3. Save and run the *16_send_rich_apple_pay_request.py* file to send the payment request.
4. Approve the payment on your test device and observe the response on your server.

Expected MSP server response

NOTE For easier understanding, the `u` has been removed from the MSP server response below.

```
Payment received!
Request Identifier: cb43a <truncated>
Payment Method Dictionary: {
  'displayName': 'Visa 8087','type': 'Debit', 'network': ’Visa'
  }

orderTracking received!
ordertracking_payload: {
  'version': '1.0',
  'payment':
   {
    'summaryItems': [{
    'amount': '1.5',
    'type':  'Final',
    'label': 'Adoption fee’
    },
    {
     'amount': '1',
     'type': 'Final',
     'label': 'Required shots’
     },
    {
     'amount': '2',
     'type': 'Final',
     'label': 'Outtake fee’
     },
    {
     'amount': '-1',
     'type': 'Final',
     'label': 'Daily discount’
     },
    {'amount': '3.5',
     'type': 'Final',
     'label': 'Your Total’
    }],
    'errors':
    []},
  'requestIdentifier': 'cb43a <truncated>'}

payload {'interactiveDataRef': { <truncated> }
```

Step 6: Send a payment request as a message

Using the code from *14_send_apple_pay_request.py*, send a payment payload for a purchase item using the merchant session.

Expected MSP server response

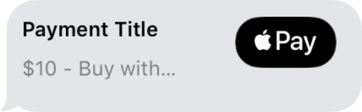
```
Business Chat server return code: 200
```

If after the payment request message was sent, it showed up as “Invalid Payment Request” in the Messages client, check the following:

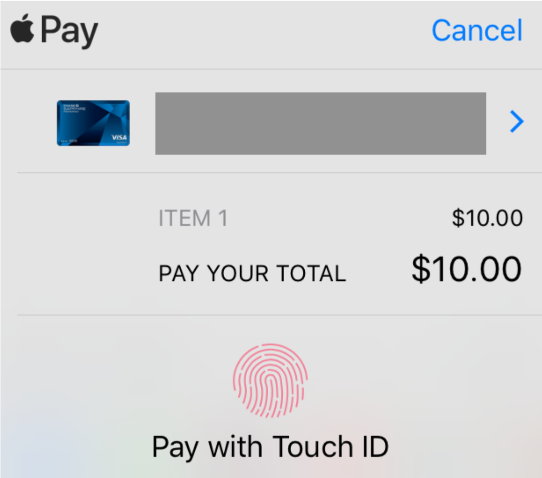
- Make sure your payment gateway URL is in the valid URL format.
- Make sure you are obtaining a new merchant session every time you are sending a payment request.

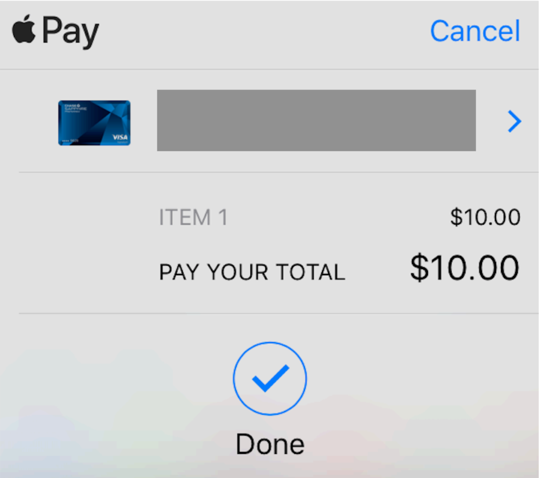
Step 7: Approve the test payment on your iPhone

You should have received the Apple Pay test payment request on your device from the previous step.



Tap the payment message to make the test payment:





Expected MSP server response

After the test payment completes, an interactive message is delivered to your endpoint with the Apple Pay payload. In the payload there is a `state` field which indicates whether the test payment was approved in your payment gateway. Be aware that you are expected to confirm the test payment transaction through your payment system, rather than relying on the state of the return message.

You should see the following output from your Payment Gateway:

```
Payment received!
Request Identifier: <request identifier in message payload>
Payment Method Dictionary: {'displayName': 'Visa 1234', 'type': 'Credit',
'network': ‘Visa'}
```

Troubleshooting tips

If you can’t receive payments in your payment gateway, check the following:

- Ensure you have a valid merchant ID, and you already verified your domain.
- Check your payment gateway is listening to POST requests on `/paymentGateway`.

If you receive a 400 error when sending the Apple Pay interactive message, make sure that you have registered your merchant ID on your account at Apple Business Register.

If your custom message doesn’t show on your client device use the following matrix identifying the type of messages sent and how they appear in the UX.

Message Type	**Reply Message**		**Response Message**	
	Text	**Picture**	**Text**	**Picture**
Apple Pay	✓	✓	✓	✓

Authentication in Business Chat

The Business Chat authentication feature allows a customer service agent or a bot to send an authorization access request to a customer for specific information.

This request displays as a bubble on the device with a Sign In call to action. The customer then signs into the authentication service using a username and password. This feature can be used for financial services, insurance, and e-commerce to ensure that customers authorize access to their private data in a way that is not visible to either the customer service provider or Apple's Business Chat servers.

There are two approaches to authentication. You can use a third-party service as the authentication service, password autofill, or you can use an authentication endpoint that you already control. For more information, see Authentication.

The Business Chat authentication feature relies on advanced cryptography. This document assumes a sufficient background in the cryptographic concepts to implement the standards described.

For readers who seek a better understanding about the field of cryptography in general, see the following:

- https://www.crypto101.io
- https://auth0.com/docs/protocols/oauth2/oauth-state

For readers who want to understand authentication and authorization technologies:

- https://www.manning.com/books/oauth-2-in-action

For readers who want to read about the standards used by our technology:

- https://oauth.net/2/
- https://tools.ietf.org/html/rfc6749#section-4.1.2

To learn how to pass authentication information using the `/authenticate` endpoint, complete the following exercises:

1. Exercise: Sending an Authentication Request.
2. Exercise: Receive and Parse an Authentication Response.
3. Exercise: Decode the Auth Token and Retrieve User Data.

Exercise: Sending an Authentication Request

For these exercises, we use a third-party service, LinkedIn's OAuth2 API, for the examples.

Before your chosen OAuth2 provider will be available, you must request a security audit for your account in Apple Business Register. This is described in more detail below. Please allow 2-3 business days for this approval.

Most third-party authentication service providers ask you to create an app on their platform. Some platforms simply refer to it as setting up a client, which is the name given under the OAuth2 spec, and some refer to it as an OAuth2 service.

For each service used, you have to tell the service where to send requests. This is called the redirect_uri or callback URL. Always enter `https://auth.businesschat.apple.com` into the field.

Prior to sending authentication requests, you need to retrieve the following fields from each third-party service and enter these into Register:

- Response type, either code or token [*]
- OAuth URL

- Token URL, for the token process; if you use theImplicit flow, leave this URL blank
- Client identifier
- Client secret
- Scope
- Decrypted token endpoint URL

PRO TIP [*] The response type "code" corresponds to Authorization Code grant type in the OAuth2 spec. The response type "token" corresponds to the Implicit grant type in the spec. These designations can lead to confusion since the "token" response type actually requires the Token URL in Apple Business Register to be blank. The authorization server provides the token directly ("implicitly") so it does not have to be retrieved by the Token URL.

The response type, OAuth URL, and token URL are from the API documentation of the authentication service provider. The client identifier and client secret are generated for you within their API account webpage.

The scope and decrypted token endpoint URLs go together, with some authentication providers offering different types of access to information. Check the API documentation for the details of which strings to use for these fields.

Step 1: Setup an Authentication Endpoint

In this first exercise, you set up an authentication service and add information about the service into Apple Business Register. There is no code to run for this task.

To follow along with these exercises, you'll need a LinkedIn developer and customer account for testing.

1. Log into LinkedIn Developer and click Create App from their API site.
2. Enter the appropriate information into the app profile on the site. Where you see Redirect URL, enter: `https://auth.businesschat.apple.com`
3. Retrieve the Client Identifier and Client Secret. These are string fields that are unique to your application, and required by Register for the LinkedIn endpoint to serve as your authentication service.
4. Go to your existing Business Chat account in Register (https://register.apple.com), find the End User Authentication section, and enter the following values:
    - OAuthURL: `https://www.linkedin.com/oauth/v2/authorization`
    - Token URL: `https://www.linkedin.com/oauth/v2/accessToken`
    - Client Identifier: Use the unique identifier from the LinkedIn API
5. Make sure you click Submit at the bottom of the account page. Changes on Register only become active after the profile is submitted and approved. The security audit for your authentication service will take 2-3 business days. Please allow sufficient time in your development schedule for this approval.
6. To ensure prompt approval, send an email reminder about the authentication audit to businesschatsupport@apple.com. Include the Business ID or a link to the URL in Apple Business Register.

Step 2: Generate Public/Private Key Pair and Nonce

This task demonstrates how to produce a unique nonce and a private/public key pair for use in the authentication request payload.

Complete this task by running the code in Listing generate_key_pair_and_nonce.py below and observe that the nonce and public/private key pairs are unique for each.

The nonce is a short string that is unique to each transaction. It does not have to be cryptographically secure to be compliant. It is a unique value for each request to ensure that a man-in-the-middle cannot perform a replay attack on the authentication provider's endpoint. To learn more about the reason for needing a unique nonce on each request, see https://auth0.com/docs/protocols/oauth2/oauth-state.

This code example below generates a random 32-character string. We could use a hash of the date and time, a sequential progression, or even a different length of string. The length of the string is explicitly left out of the spec, see https://tools.ietf.org/html/rfc6749#section-4.1.2

The key pair is formed by a cryptographically secure library. We use the hazmats methods of the Python cryptography library for the code samples. For Java, you can use BouncyCastle. Other language and platform combinations require a different library.

Use a well-tested and validated library. Many published open-source libraries may not be the best library for your platform. You can also write your own unit tests to ensure the library is compliant with the cryptography standards and security best practices that you want to build into your product.

On your server

1. Locate the code zip folder and unzip it.
2. Locate and run the *17_generate_key_pair_and_nonce.py* file.

Expected MSP server response

```
Ready to generate authentication request with parameters
request_id: 25d96 <truncated>
responseEncryptionKey: BP/Cc <truncated>
nonce (aka state): L+C55 <truncated>
private key, base64: Mxhdr <truncated>
```

Step 3: Send the Authentication Request

In this exercise, you send an authentication request to the customer’s device.

The authentication request displays as a login bubble on the device. When the customer clicks the bubble, a LinkedIn login drawer appears and the customer enters their LinkedIn username and password. After submitting the credentials, the login drawer disappears and an authentication result is displayed.

The authentication request is sent to a distinct endpoint `/authenticate`, specifically `https://mspgw.push.apple.com/v1/authenticate`. After a successful authentication, your MSP continues receiving non-authentication traffic on your `/message` endpoint.

On your server

The python code is split into two files: *auth_util.py* and *18_send_auth_request.py*. The first file, *auth_util.py*, is a library of handlers used for the exercises in the authentication section. The second file, *18_send_auth_request.py,* calls the functions in *auth_util.py* file and sends the authentication request to the customer's device.

1. Locate and edit the *18_send_auth_request.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
2. Save and run the *18_send_auth_request.py* file and ensure that the import can successfully reference the *auth_util.py* file.

Expected MSP server response

```
Save private key for use later, base 64: n9oMl <truncated>
 * Running on http://0.0.0.0:8002/ (Press CTRL+C to quit)
Business Chat server return code: 200
Send authentication request with parameters

request_id: 8c202 <truncated>
responseEncryptionKey: BGbqf <truncated>
nonce (aka state): LyZzh <truncated>
private key, base64: n9oMl <truncated>
```

Exercise: Receive and Parse an Authentication Response

In this exercise, we receive an interactive message response at our messaging endpoint with either a successful login or an unsuccessful login prompted by the authentication request.

The authentication request conforms to the interactive message class, even though it is delivered to its own endpoint. The response arrives at your CSPs `/message` endpoint, and conforms to an interactive message response.

When developing your Business Chat integration, you need to parse interactive message responses into the different types to ensure that each can be routed to the correct handler. For this exercise, we show a way of parsing the response to filter out the authentication response from other interactive message response types.

On your server

1. Locate and run the *19_receive_auth_response.py* file in background mode from the command line.
2. Ensure the library *auth_util.py* is available to the *18_send_auth_request.py* code.
3. Run the code from *18_send_auth_request.py* to generate the authentication request.
4. On the device, log in to LinkedIn.
5. On the console, observe the payload received by your MSP.

Expected MSP server response

Make a note of the encrypted token displayed in the output, this piece is used in the next exercise.

NOTE For easier understanding, the `u` has been removed from the MSP server response below.

```
payload
  {
   'interactiveData': {
      'sessionIdentifier': 'f73dd <truncated>',
      'bid': 'com.apple.messages. <truncated>',
      'data': {
         'receivedMessage': {
           'style': 'icon',
           'title': 'Sign In to LinkedIn’
           },
         'authenticate': {
           'status': 'authenticated',
           'token': 'BJuPw <truncated>'
           },
         'replyMessage': {
           'style': 'icon',
           'alternateTitle': 'You Signed In',
           'title': 'You Signed In’
           },
         'version': '1.0',
         'requestIdentifier': '8c202 <truncated>'
         }
       },
     'sourceId': 'urn:mbid:AQAAY <truncated>',
     'destinationId': '...<removed>...',
     'v': 1,
     'type': 'interactive',
     'id': 'e7ba2 <truncated>'
     }
   interactive_data: {
     'sessionIdentifier': 'f73dd <truncated>',
     'bid': 'com.apple.messages. <truncated>',
     'data': {
       'receivedMessage': {
       'style': 'icon',
       'title': 'Sign In to LinkedIn’
       },
     'authenticate': {
       'status': 'authenticated',
       'token': 'BJuPw <truncated>'
       },
     'replyMessage': {
       'style': 'icon',
       'alternateTitle': 'You Signed In',
       'title': 'You Signed In’
       },
     'version': u'1.0',
     'requestIdentifier': '8c202 <truncated>'
     }
  }

request_id: 8c202 <truncated>
status: authenticated
encrypted_token: BJuPw <truncated> #Used in Exercise: Decode the Auth Token
and Retrieve User Data
Our work is done for this routine.
```

Exercise: Decode the Auth Token and Retrieve User Data

This exercise is about decrypting the auth token and using it to request the customer information from the authentication provider.

In your full implementation, this exercise requires the copying and pasting of values delivered through a database or encrypted cloud storage. The result is a print out of the LinkedIn headline and personal information for the customer who successfully authenticates.

On your server

1. Locate and run the *19_receive_auth_response.py* file in the background.
2. Run the *18_send_auth_request.py* file and note the public and private key Base64-encodings.
3. When the authentication response is returned, the *19_receive_auth_response.py* code returns and prints the encrypted authentication token. The Base64-encoded string is notably longer than the other strings.
4. Quickly copy and paste these three strings into the *20_decode_token_get_user_data.py* file in the appropriate designated string fields. You have to work quickly since the encrypted token has an expiration—usually measured in minutes.
5. Save and run the *20_decode_token_get_user_data.py* file. It outputs the LinkedIn headline, full name, and email address for the customer who authorized the information.

We use a 30 second timeout parameter here, rather than 10 seconds in the other exercises. There is more latency in this process, so we may need to wait longer for it to complete.

Other third-party services can be similarly provisioned with the appropriate endpoint for using the decrypted auth token.

Expected MSP server response

You will see the customer’s headline and name in the payload retrieved from LinkedIn.

```
 token (decrypted): AQUqY <truncated>

 {
  'headline': 'Engineering Program Manager in Silicon Valley’,
  'lastName': 'Chen',
  'siteStandardProfileRequest': {
    'url': 'https://www.linkedin.com/profile/view?
<truncated>&authType=name&authToken=_VXR&trk=api*a5144733*s5062023*'
    },
  'id': '3FC <truncated>',
  'firstName': 'Mei'
 }
```

Connecting with Your Clients

When brands first sign up for Business Chat on Apple Business Register (ABR), they are asked to select a MSP. Your company, as a MSP, will host its own brand. Most companies, however, will subscribe to a MSP as a service from a company such as yours.

When your MSP earns a new customer, you have to have a few pieces of information from ABR. First, you have to have awareness that the new brand has chosen your MSP as a service. Second, you need the Business ID that ABR has generated for that brand. Third, you will need the brand name and logo.

These exercises cover the API that allows these data to be retrieved from ABR.

Exercise: Create a Client Landing Page

To route messages between the Business Chat server and your business clients, create a link between your clients Business Chat ID and the business client ID on your messaging platform using the *Client Landing Page URL*. When your clients access the landing page, they are presented with a login screen. The login screen logs the business client into your messaging platform and then links their Business Chat ID to a business client ID that is associated with their business name and logo.

In this step, you connect your client's Business Chat ID to your messaging platform.

1. Download and review the wireframes which are diagrams of how to set up authentication and linking between your messaging platform and your business clients.
2. Create an HTML page based on the wireframe diagrams that allows your business clients to link their Business Chat IDs to your business client ID assigned by your messaging platform.

NOTE You will save the values to your own database based on the GET handler for this page, rather than the POST handler. Most APIs use a POST for submitting information to a database. Because we are rendering a page visible to the user, this data will be handled by a GET.

This linking process allows messages to pass through your platform to their customer support agents.

3. Ensure the page is HTTPS accessible.
4. Add handlers to read the following fields from the query strings of the URL, validate the data passed, and save those items in your database: id, name, logo. These will be passed in the URL like this:

```
https://<YourSiteHost>/<YourSitePath>?
id=<BusinessChatID>&name=<BusinessName>&logo=<LogoURL>
```

5. When your clients select you as their MSP, Business Chat directs them to your landing page, with a "Connect to <Customer Service Platform>", link passing the information you need to send and receive messages for their business.
6. Automate a process to get the Business Chat ID from the URL and plug it into your messaging platform to allow your clients to test the messaging connectivity.

Listing 21_create_landing_page.py

```
from flask import Flask, render_template, request
    app = Flask(__name__)

    @app.route("/landingPage", methods=['GET'])
    def landingPage():
        return render_template("landingPage.html")
    app.run(host='0.0.0.0', port=8002)
```

Client Landing Page HTML

Create a `/templates` folder in the same directory as the HTML file. There is a sample `/templates` folder in the same directory as the other Python files in our API tutorial repository.

The following HTML is based on Jinga2 that runs parallel with Flask.

Listing templates/landingPage.html

```
    <html>
      <head>
        <title>Landing Page</title>
      </head>

      <body>
           <div class="box">
              <div class="main">
                  <h1> Welcome, {{ request.args.get('name') }}! </h1>
                  <p>Sign in to connect your Business Chat account.</p>
                    <img src={{ request.args.get('logo') }}>
                    <form class="text-center" method="post" action="/
landingPage_csp">
                        <input type="text" name="userName" placeholder="User
Name">
                        <input type="password" name="userPassword"
placeholder="Password">
                        <input type="hidden" name=“BusinessId"
value="{{ request.args.get('id') }}">
                        <input type="hidden" name="BusinessName"
value="{{ request.args.get('name') }}">
                        <input type="hidden" name="BusinessLogo"
value="{{ request.args.get('logo') }}">
                        <input type="submit" value="Get Started">
                    </form>
                    <p>Don't have an account? Click <a href=“/#”>here</a> to
create one.</p>
                    <a href="/#">Forgot password or user name?</a>
              </div>
            </div>
      </body>

      <style>
            body{text-align:center;display:flex;justify-
content:center;color:darkgray;background-color:darkgray;}
            .box{padding:2% 10%;max-width:75%;position:fixed;top:20%;border-
radius:20px;background-color:white;}
            h1 {color:black;}
            input{box-sizing: border-box;-moz-box-sizing: border-box;width:
50%;padding:10px;border-style:unset;border:1px solid darkgray;border-radius:
5px;text-align:center;}
            input[type=submit]{color:#007bff;border-color:#007bff;background-
color:initial;}
            img{max-width:200px;}
            p a {color:#007bff;text-decoration:none;}
            a {color:gray;}
      </style>
    </html>
```

Test Your Client Landing Page

To test your "Connect to <Customer Service Platform>" link, append the Business Chat ID, name, and logo to the end of your landing page URL, similar to the following:

```
https://<your-domain-here>/landingPage?id=a884eddf-
b0ad-4be4-9c0e-071531638768&name=Connect%20Business%20Chat%20Account&logo=https
://register.apple.com/assets/images/icon/apps/MessagesIcon.png
```

NOTE The "Connect to <Customer Service Platform>" link doesn't show for private CSPs.

Exercise: Publish Your Client Landing Page

In this step, you submit your company's Client Landing Page to Apple Business Register publishing it to any new customers of your messaging platform.

1. Log into your company account in Apple Business Register.
2. Find to Customer Service Platforms under Connected Services and click the "Customer Service Platforms" selection.

NOTE If you do not see "Customer Service Platforms" here, your company may not be considered a MSP in our system. Contact your Apple Business Register support representative.

3. Find your main Commercial MSP along the left navigation panel, and click it.
4. Find the "Server Configuration" Section, and then find the "Client Landing Page URL" field under this Section.

NOTE If you do not see a "Client Landing Page URL" on any of your CSPs, your company may not be have a Commercial status MSP account yet. Contact your Apple Business Register support representative.

5. Provide the landing page URL in the field indicated.
6. Click "Submit" button in the lower right corner to publish your changes to ABR.

iMessage Extensions

Business Chat provides an iMessage extension that makes it simple for you to send interactive messages to customers. But if you want to give your customers an interactive experience that is unique to your business, you can also use the Messages framework to create a standalone iMessage app or extend your iOS app so that it interacts with Messages. For more information, see iMessage App.

Exercise: Using Custom iMessage Extensions

In this exercise, you perform the following tasks:

- Set up the sample iMessage extension.
- Send an interactive message payload to trigger the sample iMessage extension with custom parameters.

Step 1: Setup a sample iMessage extension

1. Make sure you are using Xcode 9.
2. Download the extension from the Apple Developer Business Chat portal:
    - https://developer.apple.com/sample-code/wwdc/2017/iMessage-Business-Chat.zip
3. Open the Xcode project, install and run the extension on the test device.

Step 2: Send a custom iMessage extension payload

1. Copy *Development Team ID* and *Bundle Identifier* of the extension from Xcode.
2. Generate the bid parameter using prefix, team ID, and bundle identifier.
3. Assemble the extension parameter using URL parameters.
4. Assemble and send the Message payload.

On your server

1. Locate the code zip folder and unzip it.
2. Open and edit the *22_invoke_custom_extension.py* file by inserting the `destination_id`, located towards the bottom of the file, with the `source_id` from the output in Exercise: Receiving a Text Message.
3. Save and run the *22_invoke_custom_extension.py* file.

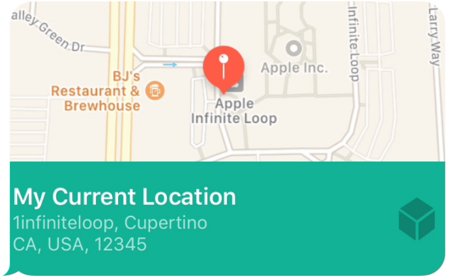
Expected MSP server response

```
Business Chat server return code: 200
```

Expected client device response

Initial Response in Messages app

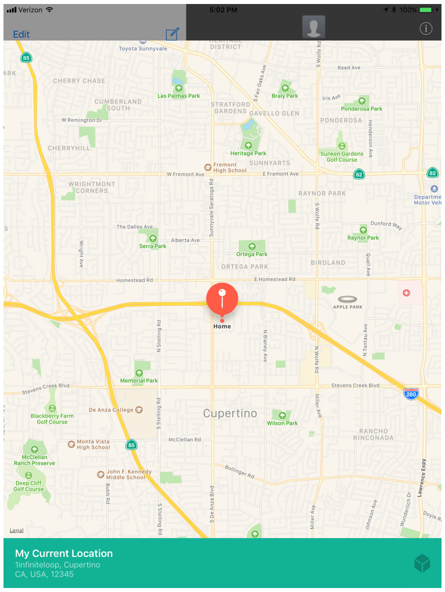
On the client device, you should see the following message bubble.



My Current Location
1infiniteloop, Cupertino
CA, USA, 12345

Message Extension UI

When the customer taps the message bubble, a detailed map opens up with a location pin.



My Current Location
1infiniteloop, Cupertino
CA, USA, 12345

Frequently Asked Questions

Most Common Error

What can I do to avoid the most common error in implementing Business Chat?

Check your encoding. Encoding errors are the largest class of technical errors based on our analysis of support tickets associated with the Business Chat API.

Often developers who have worked entirely or primarily with web technologies are unfamiliar with binary classes and objects (called byte arrays in some languages). Decoding a base64 encoded string creates a binary object. That object is not a string. String methods attempted on this object will fail.

Another common failure mode is double-encoding URLs. You should only apply safe-encoding to a URL a single time. You should use a well-validated library to URL encode and decode, to ensure that it correctly implements all aspects of the spec.

Many languages safe-encode output to log files and standard output stream. These strings may error out when used as input in the exact same format as the output. For example, Java safe-encodes many characters in strings with a / character. Python safe-encodes strings with a u' at the beginning to indicate unicode string.

Unfortunately, all valid hexadecimal strings are also valid base64 strings. When you decode a hex string as Base64, the method will succeed but create an erroneous result. This creates a difficult to triage issue.

User Accounts and Sync of Messages Across Devices

I sent my first message and it shows “Not Delivered” on the device. How do I fix it?

The “Not Delivered” error first encountered when starting Business Chat can have a number of causes. Check these items:

- Use curl command or similar to ensure that your server is listening on the correct endpoint (remember / message is appended to the end; remember it is a POST not GET)
- Ensure that the Apple ID test up on your test device is whitelisted for the account on ABR
- Ensure that the email on the test device has been registered as an Apple ID (rather than only the phone number)
- Check your server log to see whether you received the message from Business Chat and, if so, what error or HTTP response code was issuedApple

How are messages sync-ed across multiple devices for a single user?

Messages will sync all devices that are on a set of related Apple IDs. An Apple ID can be created from an email address or a phone number. You can check these associations by logging in at the Apple ID site here:

https://appleid.apple.com/

Some users set up two or more devices from different Apple IDs. The user has created, perhaps unwittingly, two zones that acts as separate users. Additional devices will sync with one or the other zone depending on the Apple ID used, but the two zones will not sync.

Business Chat conversations will have different Opaque IDs across the two zones.

What can we do to prevent issues for users with multiple devices or accounts?

Our recommended practice is to structure User tables is such a way that a single known user may have more than one Opaque ID. Most users will only enter the system once with one Opaque ID. The possibility is there, though, that they return under an unknown ID not fully realizing that is the case. This is case is more common with tech-savvy users who have many devices and many accounts for services they use, than for the general user population.

What is an Opaque ID?

The Opaque ID is referred to as the destinationId in the code samples for sending messages from your MSP to a user. It is a unique identifier to each pairing between a Business ID and a user. Each user is definite by a set of related Apple IDs as noted above.

Routing with Intent and Group IDs

How do I use the intent ID and group ID?

The intent (*biz-intent-id*) and group (*biz-group-id*) fields are used to track the source of customer engagement and to route messages to their appropriate groups.

Intent and group field values can be specified as URL parameters embedded in web sites, apps, or QR codes, or Business Chat buttons for your website. For example:

`https://bcrw.apple.com/urn:biz:Some-Biz-Chat-ID?biz-intent-id=SomeIntent&biz-group-id=SomeGroup`

In the first text message from the customer after starting the conversation through the URL, the MSP can retrieve the intent and group from the message payload. For example:

```
{
"body": "Hello",
"sourceId": "<opaque user ID>",
…
"intent": "SomeIntent",
"group": "SomeGroup"
}
```

The intent field value is specified with "biz-intent-id" key in the URL, but appears under "intent" key in the message payload. Similarly the group field is specified with "biz-group-id" in the URL and "group" from the message.

Apple Maps puts the physical address in the biz-intent-id field for users who launch Business Chat from Maps. Spaces are replaced by plus-sign characters and other safe encoding changes may be made for the sake of generating a valid URL.

For more information about routing using intent and group IDs, see About Intent, Group, and Body Values.

Compliance and Regulatory Questions

Does the MSP have to delete the conversation history with user in consideration of General Data Protection Regulation 2016/679 (GDPR)?

According to our security and compliance team:

> "The MSP is the data controller. The arrangement between the brand and the MSP guides how it is handled, with consideration of the regulatory domain they are operating in.
>
> All we offer is a block for further messages from the brand to the user. It is beyond our scope to manage the relationship any further than that."

Glossary

Term	Definition
ABC	Apple Business Chat
ABR	Apple Business Register
agent	The receiving entity for a Business Chat dialog. It can be a human customer support representative or an automated agent.
Apple ID	An Apple ID is the email address that has been preregistered at Apple: https://appleid.apple.com/. The email can be issued by an Apple domain, such as iCloud.com (http://icloud.com/) and apple.com, or a non-Apple domain.
bot	An automated agent.
business card	Business cards display the global contact details for your business, including your brand or logo, your business phone number, email, website URL, and Business Chat button. See Business and Place Cards.
business ID	Also known as a biz-ID, it is a unique business identifier assigned when Apple Register approves a Business Chat account.
console	The view of a MSP seen by customer service agents.
conversation	A long-lived interaction between a unique user Apple ID and a unique business. A conversation may consist of multiple dialogs.
CSP ID	A unique customer service platform identifier assigned when Apple Business Register approves a CSP account. Previously referred to Customer Service Provider, now called Messaging Service Provider.
CSP Secret	A unique secret retrieved from Apple Business Register along with the CSP ID.
destination ID	A field describing a messages destination. It is the user's opaque ID for messages sent to the client device or the Business ID for messages sent to the MSP. This field appears twice: once in the HTTPS header of the message as `Destination-Id` and once encrypted in the body as `destinationId`.
dialog	A set of interactions between a unique customer Apple ID and a unique bot or customer service representative that is bound in time. For example, Customer A chats with Automated Agent #1, then CSR #15, who escalated to CSR #3, who then refers Customer A back to CSR #15. This would be considered four dialogs.
group ID	A value used in URLs that designates the department or individuals best qualified to handle a customer's particular question or problem.
iMessages	Apple's secure messaging service for sending and receiving messages in the Messages app.
intent ID	A value used in URLs that define the purpose of the chat.
interactive message	A category of message within Business Chat that includes List Picker, Time Picker, Apple Pay requests, custom app download requests, and authentication requests.
Messages app	The application for sending and receiving messages. This should not be confused with iMessages.

Term	Definition
MSP	Messaging Service Platform. A messaging platform that provides the user interface for customer service agents and the connection to Business Chat or the company that sells a customer service product, such as Genesys Hub and LiveEngage by LivePerson.
opaque ID	A 169-character string that uniquely identifies the interaction between a user and a business. It is passed in the HTTP header of each Business Chat message. An opaque ID is created with each message associated with the conversation between the customer and a unique business.
link previews	A Messages app feature that Business Chat uses. The functionality of Link Previews has been found to be inconsistent, necessitating the creation of the Rich Links feature in Business Chat.
OAuth	An industry-standard protocol used for authorization. Business Chat supports OAuth 2.0.
place card	Place cards appear in iOS search results and display your business name and address. See Business and Place Cards.
POI	Point of Interest is a place of interest in Maps.
register	The portal site located at https://register.apple.com.
rich link	A MSP implemented Business Chat feature to enhance user communications. This feature ensures that media, either an image or a video, arrives on the customer's device smoothly, without presenting the "Tap to Load" message. See Sending Rich Link Messages.
source ID	A field describing where the message originates. It is the user's *opaque ID* for messages from the client device or the Business ID for messages from the MSP. This field appears twice: once in the HTTPS header of the message as `Source-Id` and once encrypted in the body as `sourceId`.