Implementation | Developer Documentation

Implementation

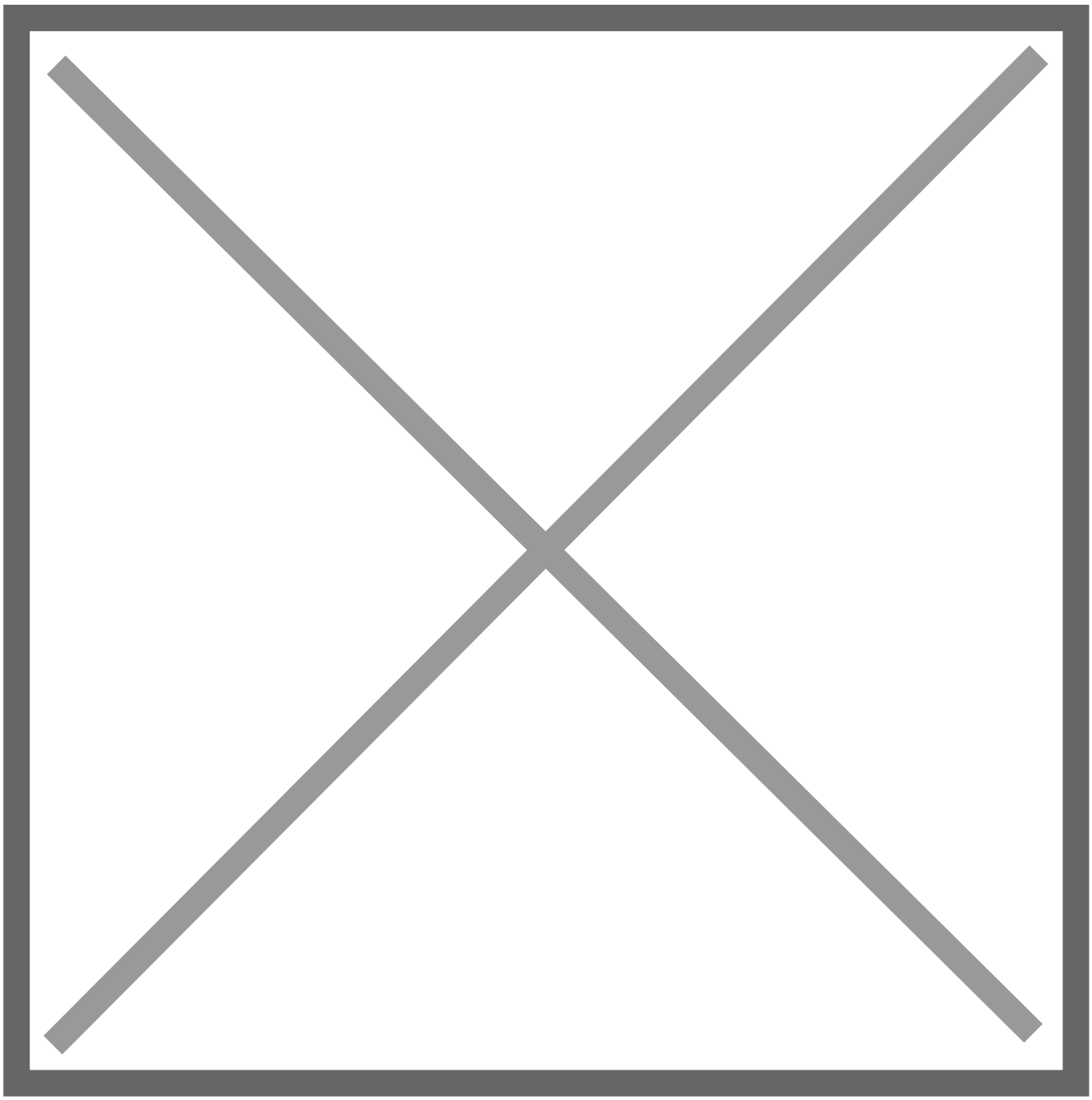
Updated: Mar 25, 2026
This document explains how to implement Embedded Signup v4 and capture the data it generates to onboard business customers onto the WhatsApp Business Platform.

Before you start

You must already be a Solution Partner or Tech Provider.If your business customers will be using your app to send and receive messages, you should already know how to use the API to send and receive messages using your own WhatsApp Business Account and business phone numbers. You should also know how to create and manage templates and have a webhooks callback endpoint properly set up to digest webhooks.You must be subscribed to the account_update webhook, as this webhook is triggered whenever a customer successfully completes the Embedded Signup flow, and contains their business information that you will need.If you are a Solution Partner, you must already have a line of credit?.The server where you will be hosting Embedded Signup must have a valid SSL certificate.

Step 1: Add allowed domains

Load your app in the App Dashboard and navigate to **Facebook Login for Business** > **Settings** > **Client OAuth settings**:



Set the following toggles to **Yes**:

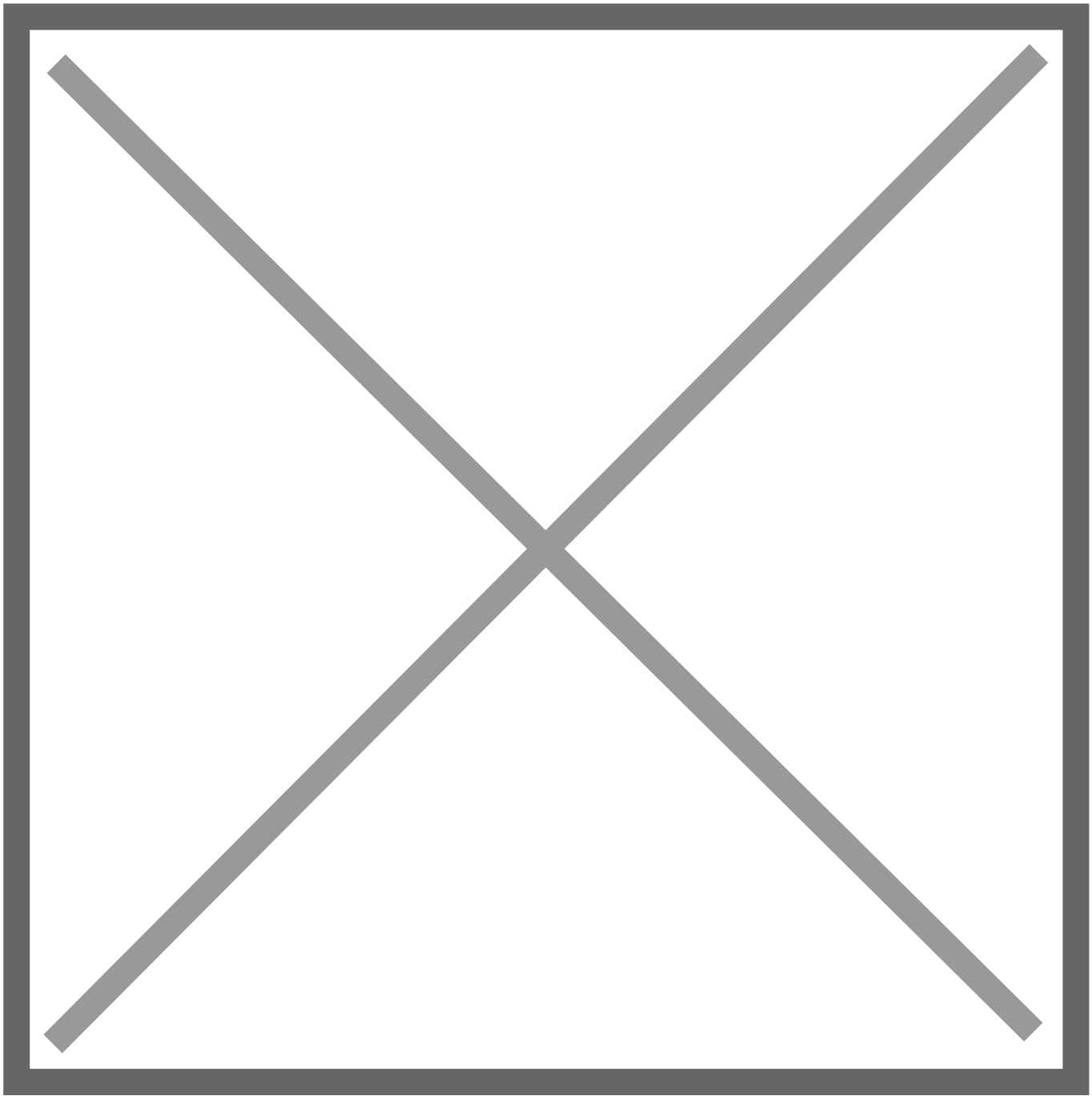
Client OAuth loginWeb OAuth loginEnforce HTTPSEmbedded Browser OAuth Loginuse Strict Mode for redirect URIsLogin with the JavaScript SDK

Embedded Signup relies on the JavaScript SDK. When a business customer completes the Embedded Signup flow, the customer's WABA ID, business phone number ID, and an exchangeable token code will be returned to the window that spawned the flow, but only if the domain of the page that spawned the flow is listed in the **Allowed domains** and **Valid OAuth redirect URIs** fields.

Add any domains where you plan to host Embedded Signup, including any development domains where you will be testing the flow, to these fields. Only domains that have enabled **HTTPS** are supported.

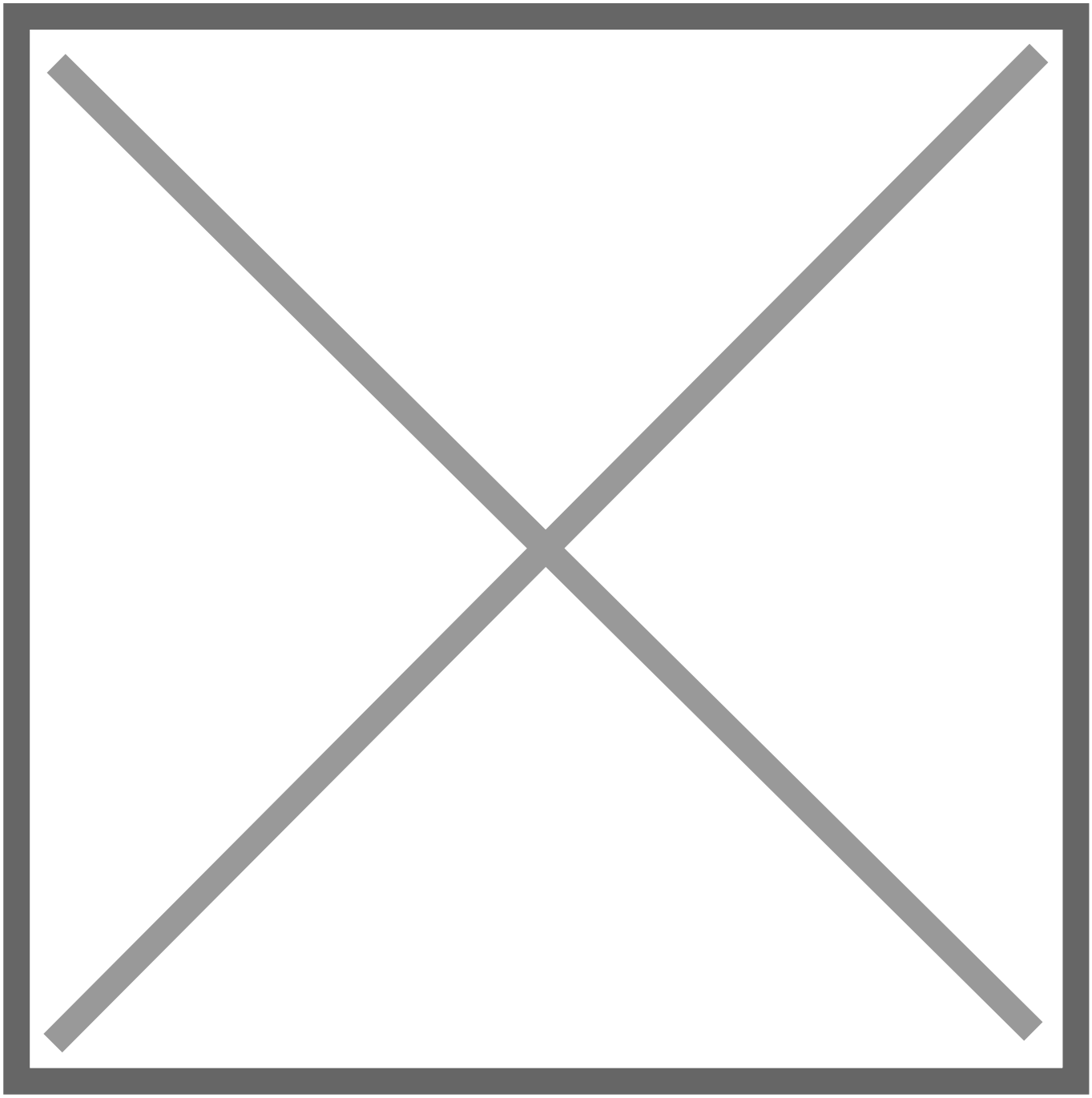
Step 2: Create a Facebook Login for Business configuration

A Facebook Login for Business configuration defines which permissions to request, and what additional information to collect, from business customers who access Embedded Signup.
Navigate to **Facebook Login for Business** > **Configurations**:

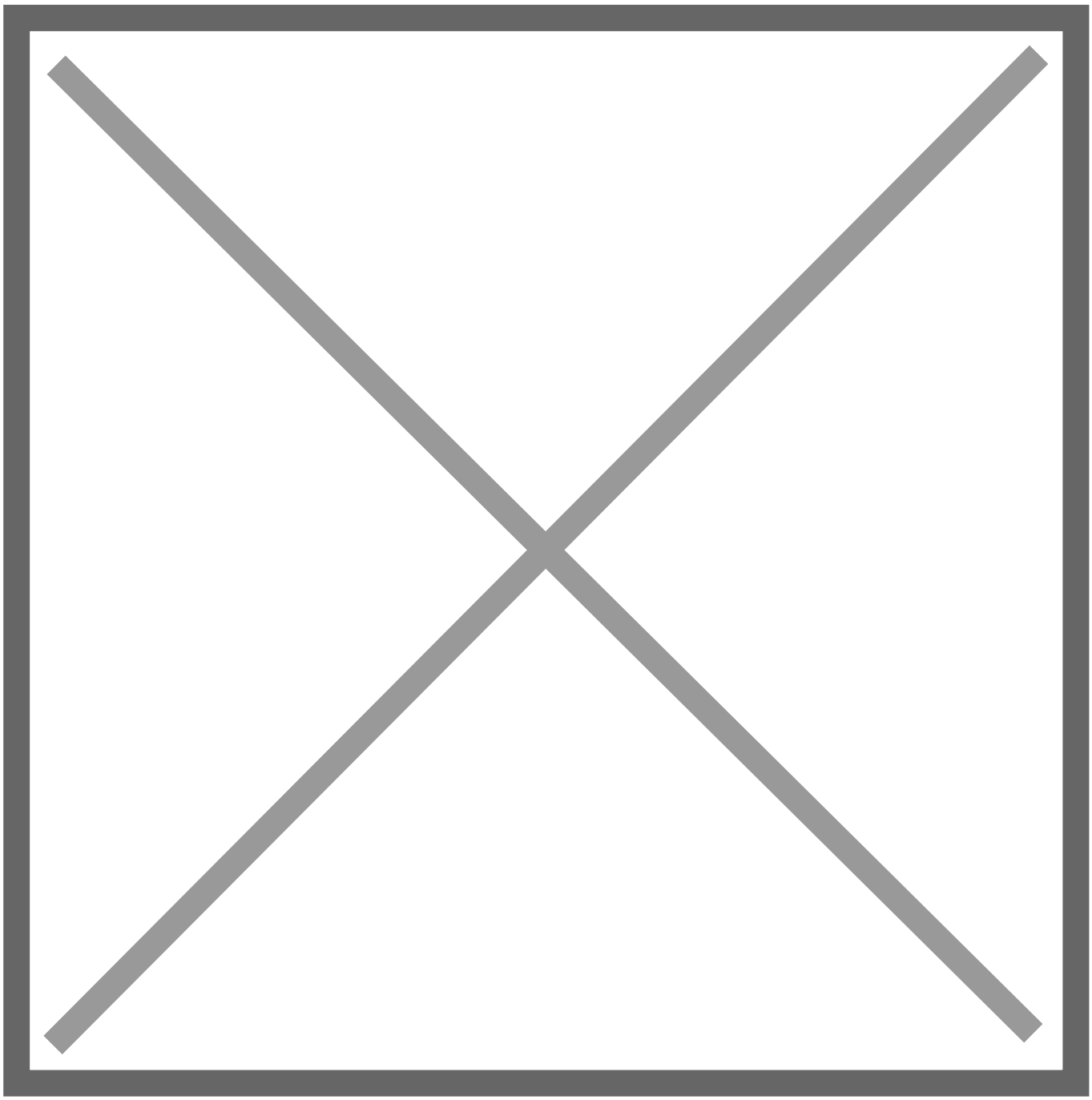


Click the **Create from template** button and create a configuration from the **WhatsApp Embedded Signup Configuration With 60 Expiration Token** template. This will generate a configuration for the most commonly used permissions and access levels.
Alternatively, you create a custom configuration. To do this, in the **Configurations** panel, click the **Create configuration** button and provide a name that will help you differentiate the custom

configuration from any others you may create in the future. When completing the flow, be sure to select the **WhatsApp Embedded Signup** login variation:

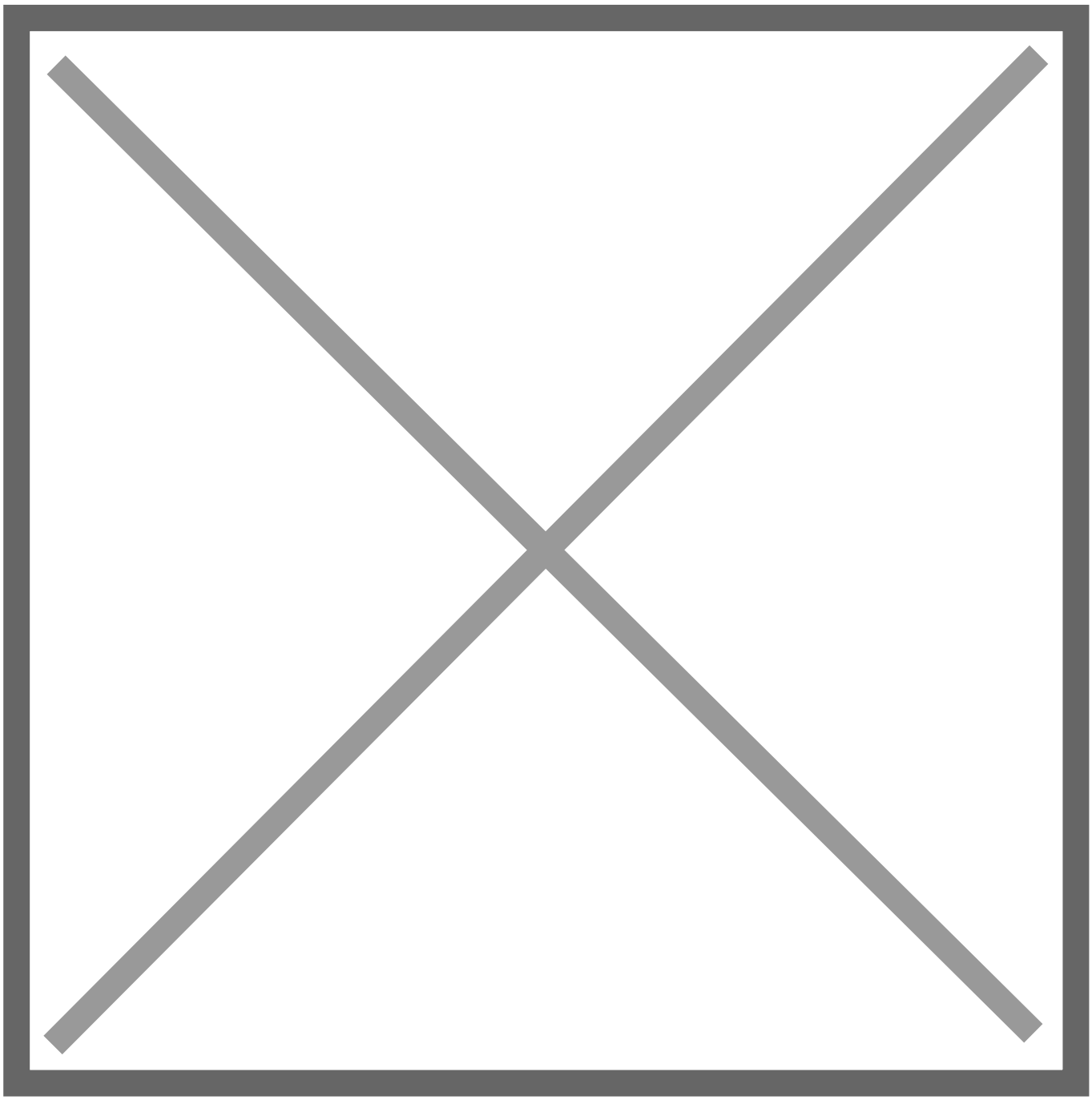


Select your products you want to onboard for this configuration.



When choosing assets and permissions, select only those assets and permissions that you will actually need from your business customers. Assets that are already selected are added by default. For example, if you select the **Catalogs** asset but don't actually need access to customer catalogs, your customers will likely abandon the flow at the catalog selection screen and ask you for clarification.

When you complete the configuration flow, capture your configuration ID, as you will need it in the next step.



Step 3: Add Embedded Signup to your website

Add the following HTML and JavaScript code to your website. This is the complete code needed to implement Embedded Signup. Each portion of the code will be explained in detail below.

Session logging message event listener

This portion of the code creates a message event listener that captures the following critical information:
The business customer's newly generated asset IDs, if they successfully completed the flowThe

name of the screen they abandoned, if they abandoned the flowAn error ID, if they encountered an error and used the flow to report it

```
// Session logging message event listener
window.addEventListener('message', (event) => {
  if (!event.origin.endsWith('facebook.com')) return;
  try {
    const data = JSON.parse(event.data);
    if (data.type === 'WA_EMBEDDED_SIGNUP') {
      console.log('message event: ', data); // remove after testing
      // your code goes here
    }
  } catch {
    console.log('message event: ', event.data); // remove after testing
    // your code goes here
  }
});
```

This information will be sent in a message event object to the window that spawned the flow and will be assigned to the data constant. **Add your own custom code to the try-catch statement that can send this object to your server.** The object structure will vary based on flow completion, abandonment, or error reporting, as described below.

Successful flow completion structure:

On the final screen, both clicking **Finish** and closing the popup (for example, by clicking the X button) are considered successful onboarding. In both scenarios, the exchangeable token code and the session info object containing the customer's asset IDs will be returned. Exiting on the final screen is not considered a cancel event.

```
{
  data: {
    phone_number_id: '<CUSTOMER_BUSINESS_PHONE_NUMBER_ID>',
    waba_id: '<CUSTOMER_WABA_ID>',
    business_id: '<CUSTOMER_BUSINESS_PORTFOLIO_ID>',

    <!-- only included if customer selected ad accounts -->
    ad_account_ids: ['<CUSTOMER_AD_ACCOUNT_ID_1>', '<CUSTOMER_AD_ACCOUNT_ID_2>'],

    <!-- only included if customer selected Facebook Pages -->
    page_ids: ['<CUSTOMER_PAGE_ID_1>', '<CUSTOMER_PAGE_ID_2>'],

    <!-- only included if customer selected datasets -->
    dataset_ids: ['<CUSTOMER_DATASET_ID_1>', '<CUSTOMER_DATASET_ID_2>'],

    <!-- only included if customer selected catalogs -->
    catalog_ids: ['<CUSTOMER_CATALOG_ID_1>', '<CUSTOMER_CATALOG_ID_2>'],

    <!-- only included if customer selected Instagram accounts -->
    instagram_account_ids: ['<CUSTOMER_IG_ACCOUNT_ID_1>', '<CUSTOMER_IG_ACCOUNT_ID_2>'],
```

```
    <!-- only included for multi-WABA flows -->
    waba_ids: ['<CUSTOMER_WABA_ID_1>', '<CUSTOMER_WABA_ID_2>']
  },
  type: 'WA_EMBEDDED_SIGNUP',
  event: '<FLOW_FINISH_TYPE>',
}
```

Placeholder	Description	Example value
`<CUSTOMER_BUSINESS_PHONE_NUMBER_ID>`	The business customer's business phone number ID	`106540352242922`
`<CUSTOMER_WABA_ID>`	The business customer's WhatsApp Business Account ID.	`524126980791429`
`<CUSTOMER_BUSINESS_PORTFOLIO_ID>`	The business customer's business portfolio ID.	`2729063490586005`
`<CUSTOMER_AD_ACCOUNT_ID>`	Only included if the customer selected ad accounts during the flow. The business customer's ad account ID.	`4052175343162067`
`<CUSTOMER_PAGE_ID>`	Only included if the customer selected Facebook Pages during the flow. The business customer's Facebook Page ID.	`1791141545170328`
`<CUSTOMER_DATASET_ID>`	Only included if the customer selected datasets during the flow. The business customer's dataset ID.	`524126980791429`
`<CUSTOMER_CATALOG_ID>`	Only included if the customer selected catalogs during the flow. The business customer's catalog ID.	`8827498273649182`
`<CUSTOMER_IG_ACCOUNT_ID>`	Only included if the customer selected Instagram accounts during the flow. The business customer's Instagram account ID.	`1749204838281942`
`<CUSTOMER_WABA_ID>` (in `waba_ids` array)	Only included for multi-WABA flows. Array of the business customer's WhatsApp Business Account IDs.	`524126980791429`

Placeholder	Description	Example value
`<FLOW_FINISH_TYPE>`	Indicates the customer successfully completed the flow. **Possible Values:** `FINISH`: Indicates successful completion of Cloud API flow. `FINISH_ONLY_WABA`: Indicates user completed flow without a phone number. `FINISH_WHATSAPP_BUSINESS_APP_ONBOARDING`: Indicates user completed flow with a WhatsApp business app number. `FINISH_OBO_MIGRATION`: Indicates user completed an on-behalf-of migration flow.`FINISH_GRANT_ONLY_API_ACCESS`: Indicates user completed a grant-only API access flow.`ERROR`: Indicates the user encountered an error during the flow.	`FINISH`

Abandoned flow structure:

```
{
  data: {
    current_step: '<CURRENT_STEP>',
  },
  type: 'WA_EMBEDDED_SIGNUP',
  event: 'CANCEL',
}
```

Placeholder	Description	Example value
`<CURRENT_STEP>`	Indicates which screen the business customer was viewing when they abandoned the flow. See Embedded Signup flow errors for a description of each step.	`PHONE_NUMBER_SETUP`

User reported errors

```
{
  data: {
    error_message: '<ERROR_MESSAGE>',
    error_code: '<ERROR_CODE>',
    session_id: '<SESSION_ID>',
    timestamp: '<TIMESTAMP>',
  },
  type: 'WA_EMBEDDED_SIGNUP',
  event: 'CANCEL',
}
```

Placeholder	Description	Example value
`<ERROR_MESSAGE>`	The error description text displayed to the business customer in the Embedded Signup flow. See Embedded Signup flow errors for a list of common errors.	Your verified name violates WhatsApp guidelines. Please edit your verified name and try again.
`<ERROR_CODE>`	Error code. Include this value if you contact support.	`524126`
`<SESSION_ID>`	Unique session ID generated by Embedded Signup. Include this ID if you contact support.	`f34b51dab5e0498`
`<TIMESTAMP>`	Unix timestamp indicating when the business customer used Embedded Signup to report the error. Include this value if you are contacting support.	`1746041036`

Parse this object on your server to extract and capture the customer's phone number ID and WABA ID, or to determine which screen they abandoned. See Abandoned flow screens for a list of possible `<CURRENT_STEP>` values and the screens they correspond to.
Note that the try-catch statement in the code above has two statements that can be used for testing purposes:

```
console.log('message event: ', data); // remove after testing


console.log('message event: ', event.data); // remove after testing
```

These statements just dump the returned phone number and WABA IDs, or the abandoned screen string, to the JavaScript console. You can leave this code in place and keep the console open to easily see what gets returned when you are testing the flow, but you should remove them when you are done testing.

Response callback

Whenever a business customer successfully completes the Embedded Signup flow, we will send an exchangeable token code in a JavaScript response? to the window that spawned the flow.

```
// Response callback
const fbLoginCallback = (response) => {
  if (response.authResponse) {
    const code = response.authResponse.code;
    console.log('response: ', code); // remove after testing
    // your code goes here
  } else {
    console.log('response: ', response); // remove after testing
```

```
    // your code goes here
  }
}
```

The callback function assigns the exchangeable token code to a `code` constant.
Add your own, custom code to the if-else statement that sends this code to your server so you can later exchange it for the customer's business token when you onboard the business customer.
Note that the if-else statement in the code above has two statements that can be used for testing purposes:

```
console.log('response: ', code); // remove after testing


console.log('response: ', response); // remove after testing
```

These statements just dump the code or the raw response to the JavaScript console. You can leave this code in place and keep the console open to easily see what gets returned when you are testing the flow, but you should remove them when you are done testing.

Launch method and callback registration

This portion of the code defines a method which can be called by an `onclick` event that registers the response callback from the previous step and launches the Embedded Signup flow.
Add your configuration ID here.

```
// Launch method and callback registration
const launchWhatsAppSignup = () => {
  FB.login(fbLoginCallback, {
    config_id: '<CONFIGURATION_ID>', // your configuration ID goes here
    response_type: 'code',
    override_default_response_type: true,
    extras: {
      setup: {},
    }
  });
}
```

Launch button

This portion of the code defines a button that calls the launch method from the previous step when clicked by the business customer.

```
<!-- Launch button -->
<button onclick="launchWhatsAppSignup()" style="background-color: #1877f2; border: 0; border-rac
```

Testing

Once you have completed all of the implementation steps above, you should be able to test the flow by simulating a business customer while using your own Meta credentials. Anyone who you have added as an admin or developer on your app (in the **App Dashboard** > **App roles** > **Roles** panel) can also begin testing the flow, using their own Meta credentials.

Onboarding business customers

Embedded Signup generates assets for your business customers, and grants your app access to those assets. However, you still need to make a series of API calls to fully onboard new business customers who have completed the flow.
The API calls you must make to onboard customers are different for Solution Partners and Tech Providers/Tech Partners.

Onboarding customers as a Solution PartnerOnboarding customers as a Tech Provider

---

Revision #5
Created 2026-04-01 15:26:28 UTC by New Admin
Updated 2026-04-06 17:51:42 UTC by New Admin