WhatsApp Integration

CONTENTS

WHY TO INTEGRATE WITH WHATSAPP?

Because it is the **most used messaging app** in the world—with over 2 billion active users each month. And because it is the best place to offer personalized customer experiences—with a click-through rate of 45–60% and an engagement rate of 98%.

But what if you want to reach those users quickly and easily? What if you want to connect with hundreds of thousands of people at scale on a platform where they're most active?

That's where **WhatsApp API providers** come in.

With an API provider, you can **broadcast messages to unlimited users**. You can also **automate notifications**, **integrate chatbots**, **offer live chat** on many platforms, and do much more with WhatsApp API.

IS THERE A DIRECT CONNECTION FOR INTEGRATING WHATSAPP WITH BUSINESS CENTRAL?

There isn't a direct integration between WhatsApp and Microsoft Dynamics 365 Business Central out of the box.

Use a Third-Party Integration Platform:

- There are third-party integration platforms and tools that can help connect different applications, including WhatsApp and Business Central.

List of Top WhatsApp API Providers :

- Twilio
- WATI
- Interakt
- MessageBird
- Morph
- Zoko
- 360Dialog
- TextLocal
- Clickatell
- Yellow.ai

NOTE1: **But here we are using Meta for integration.**

WHAT IS META PLATFORM?

Meta Platforms doing business as Meta and formerly named Facebook. The company owns and operates Facebook, Instagram, Threads, and WhatsApp, among other products and services.

META APP DEVELOPMENT

Before you can integrate with Meta's products, use Meta's SDKs, or access any of Meta's APIs, first register as a Meta developer. After you register, use the app dashboard to provide information about your app. This documentation explains how to register as a developer, how to use the app dashboard to configure your app, and how to build, test, and release your app.

1.REGISTER AS A META DEVELOPER

This document explains how to register as a Meta developer. Once you have registered you will have access to the App Dashboard and all of our products, SDKs, APIs, development tools, and developer documentation.

Step 1: Start the registration process.

While logged into your Facebook account, go to https://developers.facebook.com/async/registration.

Alternatively, you can go to the Meta for Developers website and click **Get Started**.

Step 2: Agree to our Terms and Policies

Click **Next** to agree to our Platform Terms and Developer Policies.

Step 3: Verify your account.

We will send a confirmation code to the phone number and email address that you provide in order to confirm that you have access to them. Your number and email will be used for important developer notifications of any changes that may impact to your app.

Step 4: Select your occupation.

Select an occupation that most closely describes what you do for a living.

Next Steps

Now that you have registered you can use the App Dashboard to create your first app.

2.CREATE AN APP

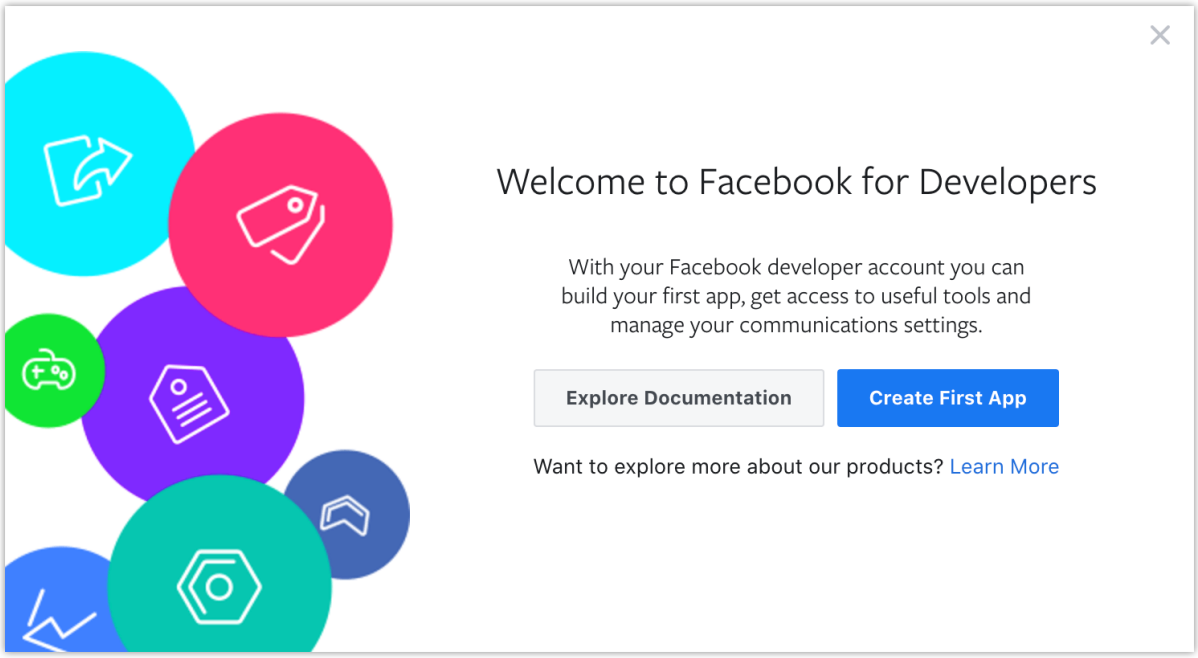
The app creation flow gathers the minimum amount of information needed to generate a unique ID for your app. Once you complete the flow you will end up in the App Dashboard where you can provide additional information about your app, or start building and testing right away.

Before You Start

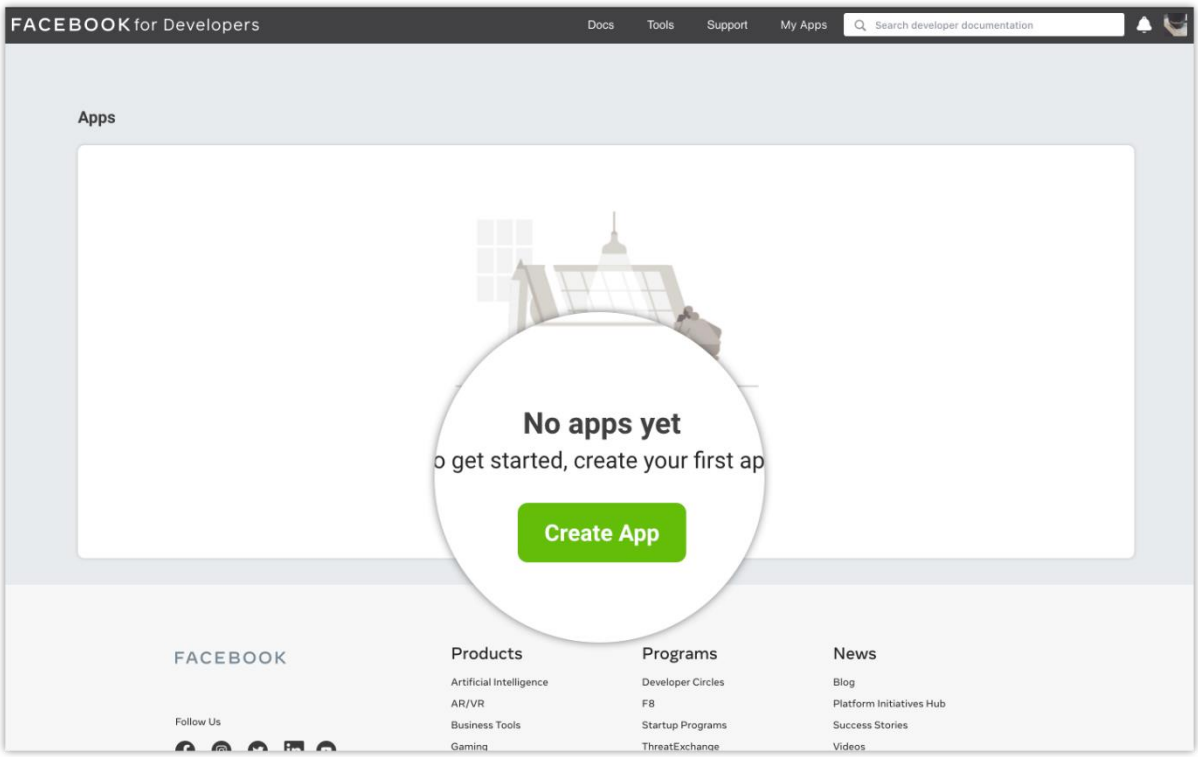
- You must be logged into your Meta developer account.
- Ensure you have not exceeded your app limit. As a Developer, you are permitted to have a developer or administrator role on a maximum of 15 apps. This limit applies to your apps not already linked to a Meta Verified Business. If you have reached the app limit and are unable to create an application or accept a new pending role, take the following steps:
    - Visit your My Apps page to remove any apps you no longer use. You can remove an app entirely or resign your role as an administrator or developer from the app.
        - Archived apps count towards the app limit; if you no longer require these apps, we suggest removing them.
        - If you have already gone through Meta Business Verification, link any unlinked applications to that verified business.
            - If you have not gone through Meta Business Verification and are eligible, please follow the steps to get your business verified.
            - To check if your app is linked to a verified business, check under Basic Settings -> Verifications -> Business verification.

Step 1: Start the app creation process.

If you just came from the registration flow, click the **Create First App** button.



Otherwise, go to the Apps panel and click **Create App**.



Step 2: Choose a use case.

Your use case determines which permissions, products and APIs are available to your app.

Select a use case and click **Next**.

If you would like to add Facebook Login for Business, as well as create apps to track ads and manage your business, select the other option when creating your app.

If you select **Other**, meaning you want to implement ads, gaming, or messaging, perhaps in addition to Facebook Login, you'll be redirected to our App Types creation flow (see below).

Step 3: Set your app name and email.

Enter the name of your app and an email address where we can send you any important developer notifications. The email address can be different from the email address associated with your Facebook account, just make sure it's valid and that you monitor it, since all important developer notifications will be sent there. You can also add a Meta Business Account if you have one, then click **Create App**.

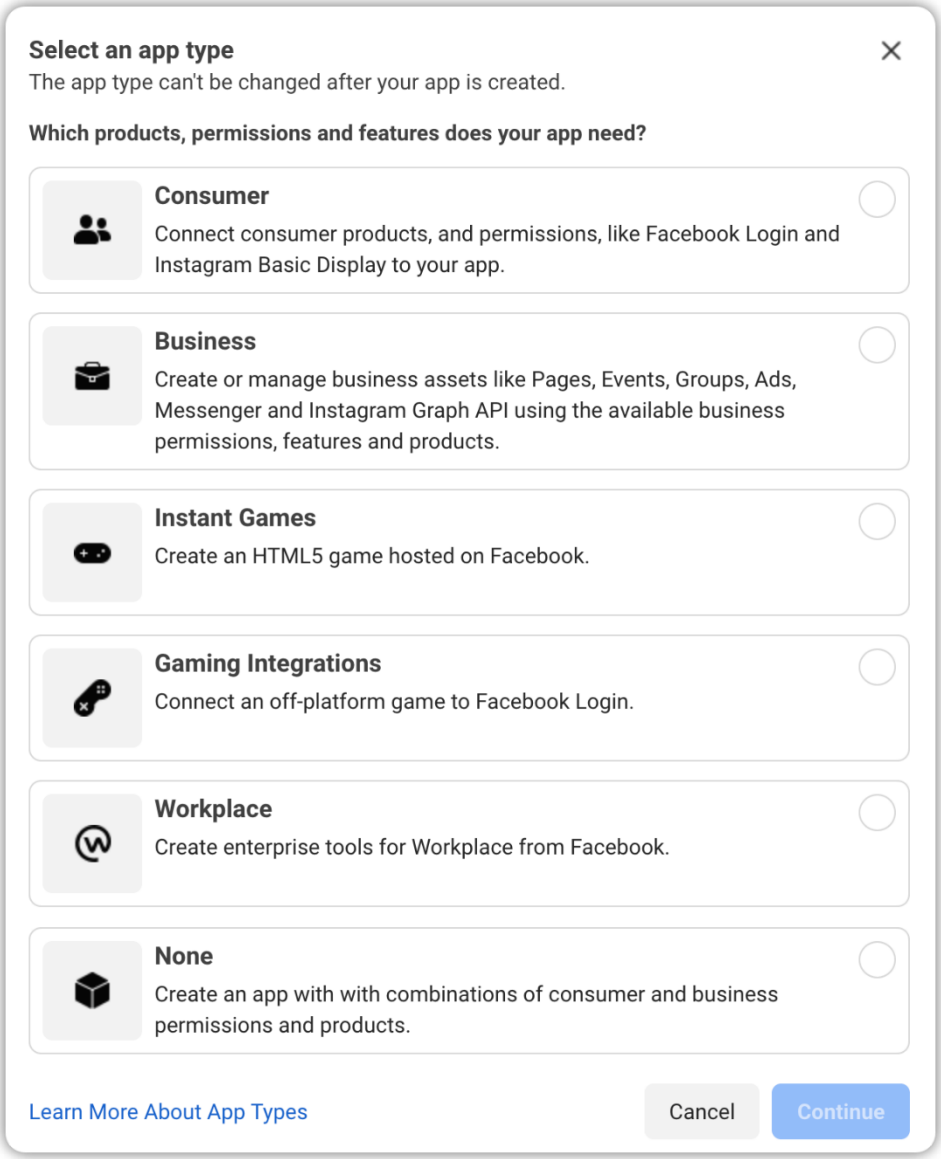
Once you have completed the app creation flow, your app will be loaded in your app dashboard. The app dashboard allows you to view, add, and update details, such as app settings, roles, and additional use cases. You can also request access to the permissions your app will need to make successful API calls to Graph API endpoints for your use cases.

App Type Creation Flow

If you need to implement more products, such as ads, games, or messaging, in addition to Facebook Login or you do not need Facebook Login, you will select an app type during the app creation flow instead of a use case.

Step 1. Choose an app type.

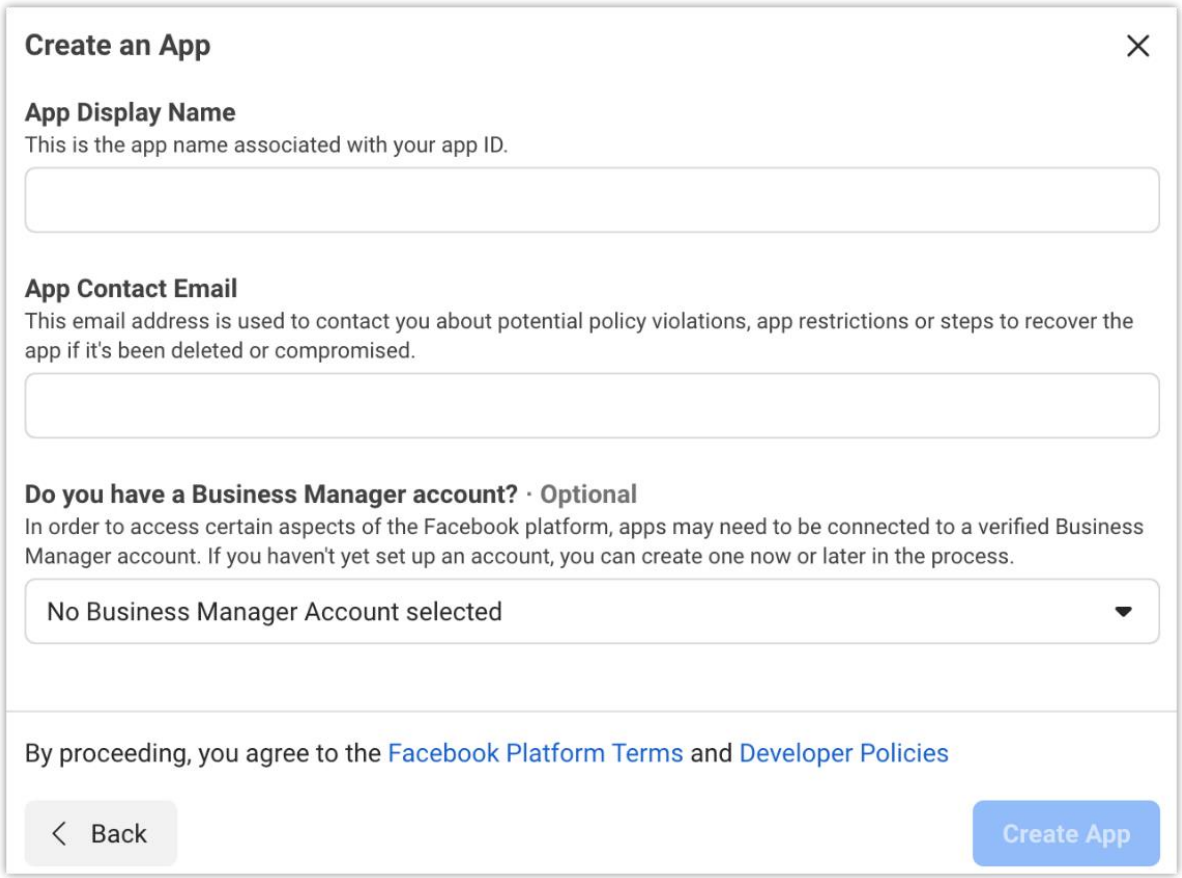
Your app type determines which products and APIs are available to your app. If this is your first time creating an app and you are just exploring the app creation flow, choose the **None** option. Later, when you are more familiar with our products and APIs, refer to our app types document to determine which app type is best suited for your app, then create a new app and choose an appropriate type.



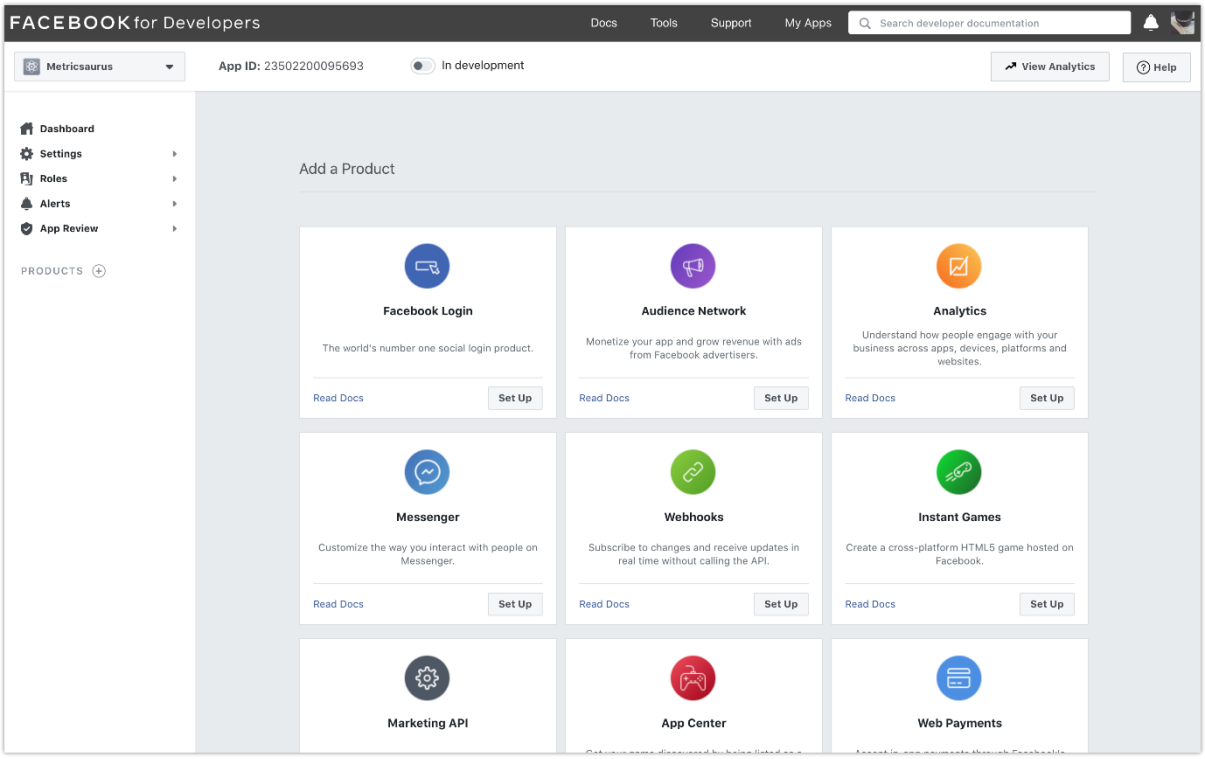
Step 2. Set your app name and email

Enter the name of your app and an email address where we can send you any important developer notifications. The email address can be different from the email address associated with your Facebook account, just make sure it's valid and that you monitor it, since all important developer notifications will be sent there.

You can also add a Meta Business Manager account if you have one, then click Create App.



Once you have completed the app creation flow your app will be loaded in the App Dashboard.



APP DASHBOARD

This document describes the App Dashboard interface and its settings. **If you have just created a new app, you do not need to read all of this documentation and configure dashboard settings now**; you can start building and testing right away and return to these documents to learn about relevant dashboard settings as needed.

The App Dashboard allows you to configure settings that may be required by the use cases, APIs, and SDKs that your app will be using. It also provides tools to aid with app development, such as API usage meters, the ability to create test users and test pages, and the ability to assign roles to other people who may be helping you with development. The dashboard is also used to begin the App Review, Business Verification process, if required.

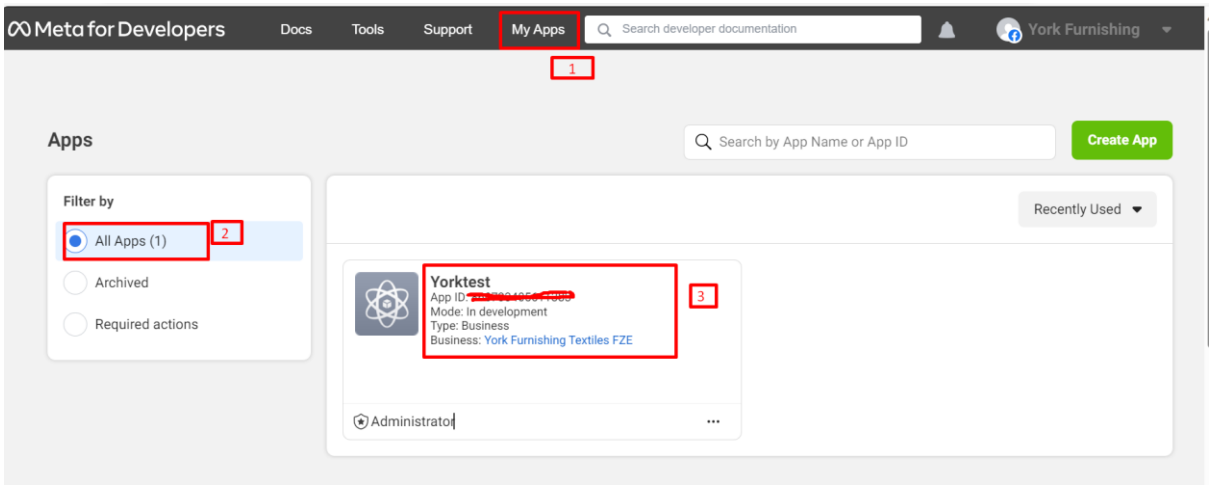
To access the App Dashboard, go to My Apps and click on the name of the app you'd like to configure. This will load the app in the dashboard, and you can then adjust its settings.

MY APPS:

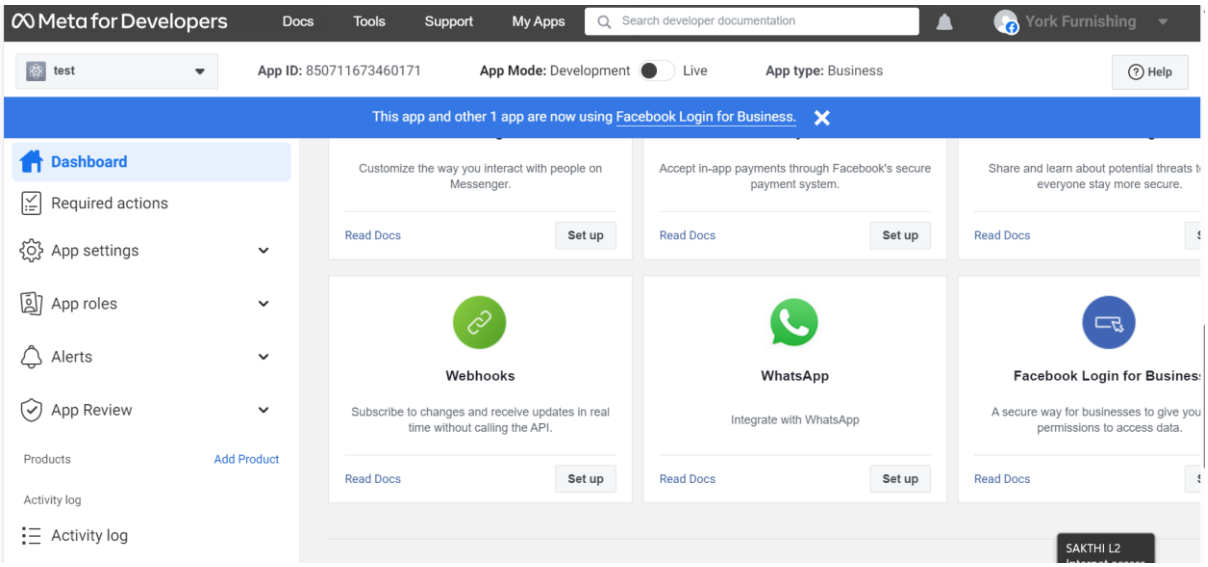
My Apps serves as the App Dashboard's entry point. It displays basic information about each of your apps: their IDs, your role on each of them, and the number of unread developer notifications they have received. It also allows you to remove yourself from apps that you have been invited to by other developers, remove your own apps, archive your own apps, and begin the Data Use Checkup process, if required.

You can access the **Apps** page at developers.facebook.com/apps or by clicking the **My Apps** link in the header that appears on all Meta for Developers Documentation pages.

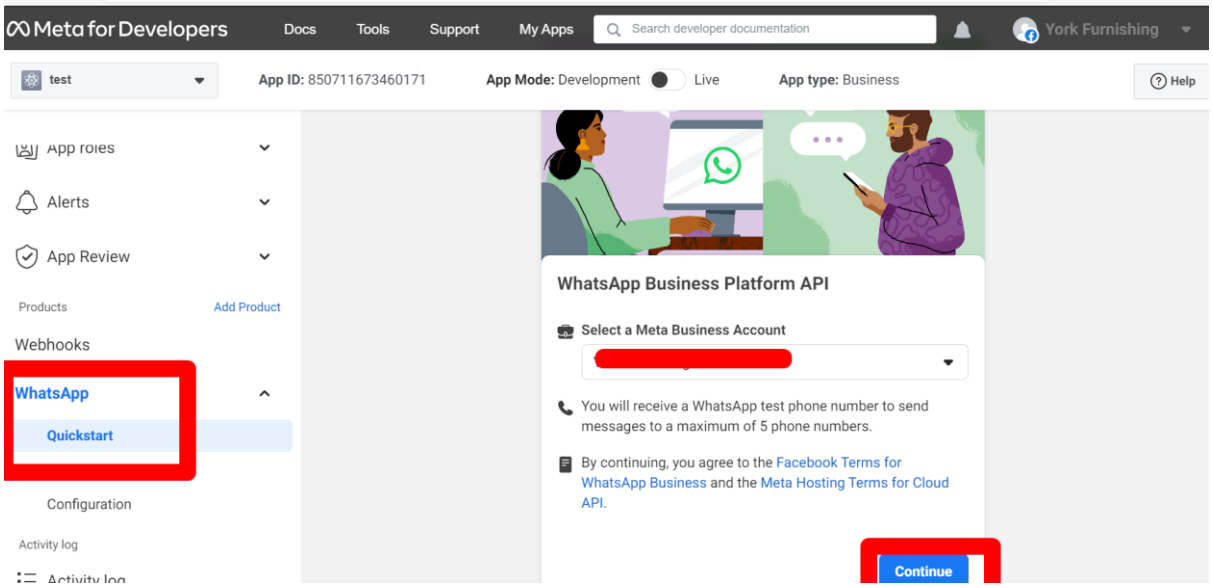
You can view like below:



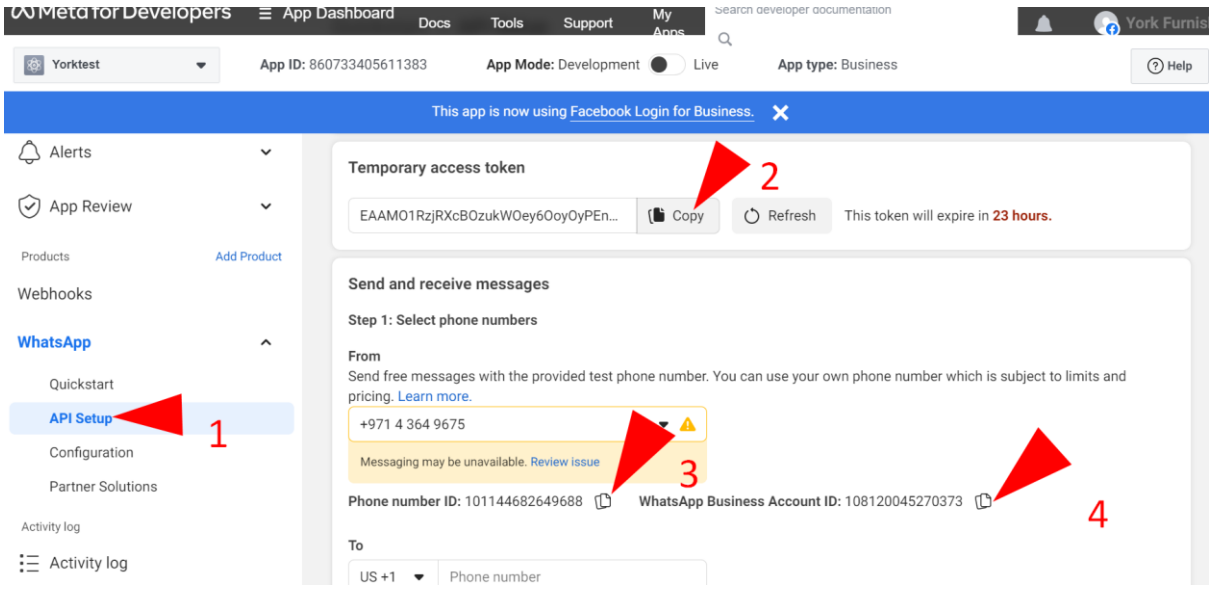
When open the app, you can view the dashboard. In that dashboard, you can setup the WhatsApp.

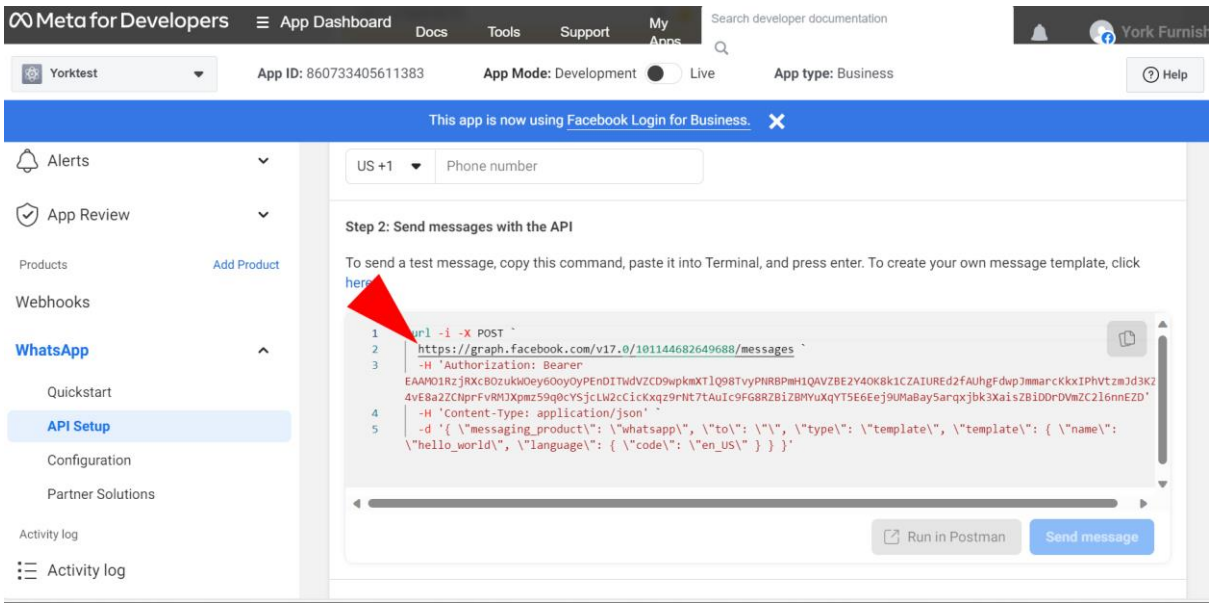


Go to WhatsApp Setup and select a Meta Business Account. Then click on Continue.



In the below screen you will get a detail regarding the WhatsApp Integration.





In the above screen you will get the WhatsApp Integration API.

Those are the steps are done by Customer side. And they need to add the Payment method.

REQUIRED INFORMATION FOR WHATSAPP INTEGRATION DEVELOPMENT:

- WhatsApp API
- Phone Number ID
- Bearer Access Token

WHAT IS WHATSAPP BUSINESS PLATFORM?

The WhatsApp Business Platform gives medium to large businesses the ability to connect with customers at scale. You can start conversations with customers in minutes, send customer care notifications or purchase updates, offer your customers a level of personalized service and provide support in the channel that your customers prefer to be reached on.

The WhatsApp Business Platform consists of the following APIs:

WhatsApp Business Platform Cloud API

The Cloud API allows you to send and receive messages to and from customers using cloud-based servers owned by Meta. Since we host the API, you avoid the cost of hosting your own servers and can easily scale your business messaging.

WhatsApp Business Platform On-Premises API

The On-Premises API allow you to send and receive messages to and from customers using your own servers.

WhatsApp Business Management API

The WhatsApp Business Management API allows you to manage your WhatsApp Business Account settings & assets and get quality status updates.

NOTE2: We are using WhatsApp Business Platform Cloud API here.

GET STARTED..

This guide helps you get started with Cloud API and is intended for people developing for themselves or their organization, not on behalf of a client. If you're developing on behalf of a client, see this form. All developers must follow our WhatsApp's Commerce Policy.

To send and receive a first message using a test number, complete the following steps:

1. Set up developer assets and platform access
2. Send a test message
3. Configure a Webhook
4. Receive a test message

Once you're ready to use your app for a production use case, check the Next Steps section.

1. SET UP DEVELOPER ASSETS AND PLATFORM ACCESS

The Cloud API and Business Management API are part of Meta's Graph API, so you need to set up a Meta developer account and a Meta developer app. To set that up:

- Register as a Meta Developer
- Enable two-factor authentication for your account
- Create a Meta App: Go to **developers.facebook.com > My Apps > Create App**. Select the **Business** type and follow the prompts on your screen. If you are asked to choose a **Use Case** as part of the app creation flow, choose **Other** as your use case, then select **Business**.

From the App Dashboard, click on the app you would like to connect to WhatsApp. Scroll down to find the "WhatsApp" product and click **Set up**.

Next, you will see the option to select an existing Business Manager (if you have one) or, if you would like, the onboarding process can create one automatically for you (you can customize your business later, if needed).

Make a selection and click **Continue**. This will:

1. Associate your app with the Business Manager account that you selected earlier (or had created for you).

2. Generate a WhatsApp Business Account.
3. Generate a test business phone number and associate it with your WhatsApp Business Account. You can use this number with the API to send an unlimited number of messages to up to 5 recipient phone numbers. Recipient phone numbers can be any valid number, but you must verify each one in the next step.
4. Redirect you to the **WhatsApp > API Setup** panel in the App Dashboard.

2. SEND A TEST MESSAGE

In the **WhatsApp > API Setup** panel:

1. Select your test phone number in the **From** field.
2. Enter the recipient phone number you would like to message in the **To** field. Ensure the number is correct, and that you want to add it to your list of 5 possible message recipients —as you add phone numbers, follow the prompts on the screen to verify you have access to them. **Once this number has been added, it cannot be removed from your list.** Note: This limitation is only for WhatsApp-provided test phone numbers. Real phone numbers that you register do not have a limit on the number of recipients.
3. Once you enter a recipient phone number, the code sample on the page will be updated to demonstrate an API call that sends a pre-approved message template to that number. Message templates are the only type of message that can be sent to customers who have yet to message you or have not messaged you in the last 24 hours. Thus, message templates are primarily used to open marketing, utility, and authentication conversations with customers.

The updated code sample will look something like this:

```
curl -i -X POST \

  https://graph.facebook.com/v18.0/105954558954427/messages \

  -H 'Authorization: Bearer EAAFl...' \

  -H 'Content-Type: application/json' \

  -d '{ "messaging_product": "whatsapp", "to": "15555555555", "type":
"template", "template": { "name": "hello_world", "language": { "code": "en_US"
} } }'
```

1. Finally, click **Send message** to send the first message. As an alternative, you can copy the code sample provided and execute it in your Terminal or in Postman. **You have just sent a test message!**

The code sample on the page is formatted for use in Unix-style terminal shells and is expected to work on MacOS and distributions of Gnu/Linux. If you use Windows, we suggest you perform your first API call using Postman, to avoid platform-related URL formatting concerns. If you are a Windows 10 user, URL is available, but requires a different syntax than the one shown in the panel to execute in PowerShell or cmd.exe. For more information, see URL Comes to Windows or URL for Windows. If you have access to the Windows Subsystem for Linux (WSL), you can also consider launching a Linux distribution and using its terminal.

3. CONFIGURE A WEBHOOK

To get alerted when you receive a message or when a message's status has changed, you need to set up a Webhooks endpoint for your app. Setting up Webhooks doesn't affect the status of your phone number and does not interfere with you sending or receiving messages.

To get started, first you need to create the endpoint. You can create a custom Webhook URL running on a web server, or use services that help you set up an endpoint, such as Glitch. See Create a Sample App Endpoint for Webhooks Testing for help.

Once your endpoint is ready, go to your App Dashboard.

In the App Dashboard, go to **WhatsApp > Configuration**, then click the **Edit** button.

- Callback URL: This is the URL Meta will be sending the events to. See the Webhooks, Getting Started guide for information on creating the URL.
- Verify Token: This string is set up by you, when you create your webhook endpoint.

After adding the information, click **Verify and Save**.

After saving, back in the **Configuration** panel, click the **Manage** button and subscribe to individual webhook fields. To receive notifications of customer messages, be sure to subscribe to the **messages** webhook field.

4. RECEIVE A TEST MESSAGE

Now that your Webhook is set up, send a message to the test number you have used. You should immediately get a Webhooks notification with the content of your message!

NEXT STEPS

PHONE NUMBER

When you're ready to use your app for a production use case, you need to use your own phone number to send messages to your users. When choosing a phone number, consider the following:

- If you want to use a number that is already being used in the WhatsApp customer or business app, you will have to fully migrate that number to the business platform. Once the number is migrated, you will lose access to the WhatsApp customer or business app. See Migrate Existing WhatsApp Number to a Business Account for information.
- We have a set of rules regarding numbers that can be used in the platform. Learn more.
- Once you have chosen your phone number, you have to add it to your WhatsApp Business Account. See Add a Phone Number.

OPT-IN

You are required to obtain user opt-in before opening marketing, utility, and authentication conversations with customers.

PRICING & PAYMENT METHODS

Businesses are charged per conversation, which includes all messages delivered in a 24 hour session. The first 1,000 Service conversations each month are free. If you want to have more than 1,000 Service conversations, you must add a credit card to your account.

If you're a developer who is developing for yourself or your organization, not on behalf of a client, you may add a credit card to your WhatsApp Business account. To set up a credit card as a payment method, go to **App Dashboard > WhatsApp > Configuration**. Under Phone numbers, click **Manage phone numbers**. This leads you to your Business Manager account, where you can click **Add Payment Method** to add your credit card. Currently only Visa and Mastercard are supported.

There are six currencies available for payment: USD, AUD, EUR, GBP, IDR, INR. Credit card payment method is available if you're located in any of the countries listed in the Business Manager Supported Countries for WhatsApp Business Credit Card Billing help center article.

For more information, see Add a Credit Card to Your WhatsApp Business Platform Account.

SCALE CONVERSATIONS

To be able to open more marketing, utility, and authentication conversations, you need to use message templates. WhatsApp message templates are specific message formats that businesses use to send out notifications or customer care messages to people that have opted in to notifications.

Before sending a message template, you need to create one or you can use one of our pre-approved templates. Check this guide to learn how to send message templates.

Incoming messages are unlimited, but there are limits for outgoing messages. See Messaging Limits for more information on messaging tiers.

CONSTRAINTS

- The following types of message are supported: text, message templates, images, documents and audio.
- A text message can be a max of 4096 characters long.

CREATING

You send messages by making a POST call to the /messages node regardless of message type. The content of the JSON message body differs for each type of message (text, image, etc.).

PARAMETERS

These are the main parameters used in /messages POST requests:

Name	**Description *(Click the arrow in the left column for supported options.)***
audio *object*	**Required when type=audio.** A media object containing audio.
contacts *object*	**Required when type=contacts.** A contacts object.
context *object*	**Required if replying to any message in the conversation.** An object containing the ID of a previous message you are replying to. For example: {"message_id":"MESSAGE_ID"} *Cloud API only.*
document	**Required when type=document.**

Name	Description *(Click the arrow in the left column for supported options.)*
object	A media object containing a document.
hsm *object*	Contains an hsm object. This option was deprecated with v2.39 of the On-Premises API. Use the template object instead. *On-Premises API only.*
image *object*	**Required when type=image.** A media object containing an image.
interactive *object*	**Required when type=interactive.** An interactive object. The components of each interactive object generally follow a consistent pattern: header, body, footer, and action.
location *object*	**Required when type=location.** A location object.
messaging_product *string*	**Required** Messaging service used for the request. Use "whatsapp". *Cloud API only.*
preview_url	**Required if type=text.**

Name	Description *(Click the arrow in the left column for supported options.)*
boolean	Allows for URL previews in text messages — See the Sending URLs in Text Messages. This field is optional if not including a URL in your message. **Values:** false (default), true. *On-Premises API only. Cloud API users can use the same functionality with the preview_url field inside a text object.*
recipient_type *string*	**Optional.** Currently, you can only send messages to individuals. Set this as individual. Default: individual
status *string*	A message's status. You can use this field to mark a message as read. See the following guides for information: • Cloud API: Mark Messages as Read • On-Premises API: Mark Messages as Read
sticker *object*	**Required when type=sticker.** A media object containing a sticker. **Cloud API**: Static and animated third-party outbound stickers are supported in addition to all types of inbound stickers. A static sticker needs to

Name	Description *(Click the arrow in the left column for supported options.)*
	be 512x512 pixels and cannot exceed 100 KB. An animated sticker must be 512x512 pixels and cannot exceed 500 KB. **On-Premises API**: Only static third-party outbound stickers are supported in addition to all types of inbound stickers. A static sticker needs to be 512x512 pixels and cannot exceed 100 KB. Animated stickers are not supported.
template *object*	**Required when type=template.** A template object.
text *object*	**Required for text messages.** A text object.
to *string*	**Required.** WhatsApp ID or phone number of the customer you want to send a message to. See Phone Number Formats. If needed, On-Premises API users can get this number by calling the contacts endpoint.
type *string*	**Optional.** The type of message you want to send. If omitted, defaults to text.

TEXT OBJECT

Name	Description
body	**Required.** Contains the text of the message, which can contain URLs and formatting.

MEDIA OBJECT

For the On-Premises API, the media object id is returned when the media is successfully uploaded to the WhatsApp Business on-premises/reference client via the media endpoint.

Name	Description
id *string*	**Required** **when type is audio, document, image, sticker, or video and you are not using a link.** The media object ID. Do not use this field when message type is set to text.
link *string*	**Required** **when type is audio, document, image, sticker, or video and you are not using an uploaded media ID (i.e. you are hosting the media asset on your public server).** The protocol and URL of the media to be sent. Use **only** with HTTP/HTTPS URLs. Do not use this field when message type is set to text.

Name	Description
	Cloud API users only: • See Media HTTP Caching if you would like us to cache the media asset for future messages. • When we request the media asset from your server you must indicate the media's MIME type by including the Content-Type HTTP header. For example: Content-Type: video/mp4. See Supported Media Types for a list of supported media and their MIME types.
caption *string*	**Optional.** Media asset caption. Do not use with audio or sticker media. **On-Premises API users:** • For v2.41.2 or newer, this field is is limited to 1024 characters. • Captions are currently not supported for document media.
filename *string*	**Optional.** Describes the filename for the specific document. Use **only** with document media.

Name	Description
	The extension of the filename will specify what format the document is displayed as in WhatsApp.
provider *string*	**Optional. On-Premises API only.** This path is optionally used with a link when the HTTP/HTTPS link is not directly accessible and requires additional configurations like a bearer token. For information on configuring providers, see the Media Providers documentation.

CONTACTS OBJECT

Inside contacts, you can nest the following objects: addresses, emails, name, org, phone, and urls. **Pluralized objects are to be wrapped in an array as shown in the example below.**

Name	Description
addresses *object*	**Optional.** Full contact address(es) formatted as an addresses object. The object can contain the following fields: street*string* – **Optional.** Street number and name. city*string* – **Optional.** City name. state*string* – **Optional.** State abbreviation. zip*string* – **Optional.** ZIP code. country*string* – **Optional.** Full country name. country_code*string* – **Optional.** Two-letter country abbreviation.

Name	Description
	type*string* – **Optional.** Standard values are HOME and WORK.
birthday	**Optional.** YYYY-MM-DD formatted string.
emails *object*	**Optional.** Contact email address(es) formatted as an emails object. The object can contain the following fields: email*string* – **Optional.** Email address. type*string* – **Optional.** Standard values are HOME and WORK.
name *object*	**Required.** Full contact name formatted as a name object. The object can contain the following fields: formatted_name*string* – **Required.** Full name, as it normally appears. first_name*string* – **Optional*.** First name. last_name*string* – **Optional*.** Last name. middle_name*string* – **Optional*.** Middle name. suffix*string* – **Optional*.** Name suffix. prefix*string* – **Optional*.** Name prefix. *At least one of the optional parameters needs to be included along with the formatted_name parameter.

Name	Description
org *object*	**Optional.** Contact organization information formatted as an org object. The object can contain the following fields: company*string* – **Optional.** Name of the contact's company. department*string* – **Optional.** Name of the contact's department. title*string* – **Optional.** Contact's business title.
phones *object*	**Optional.** Contact phone number(s) formatted as a phone object. The object can contain the following fields: phone*string* – **Optional.** Automatically populated with the `wa_id` value as a formatted phone number. type*string* – **Optional.** Standard Values are CELL, MAIN, IPHONE, HOME, and WORK. wa_id*string* – **Optional.** WhatsApp ID.
urls *object*	**Optional.** Contact URL(s) formatted as a urls object. The object can contain the following fields: url*string* – **Optional.** URL. type*string* – **Optional.** Standard values are HOME and WORK.

Example of a contacts object with pluralized objects nested inside:

```
"contacts": [
  {
    "addresses": [
      {
        "city": "city name",
        "country": "country name",
        "country_code": "code",
        "state": "Contact's State",
        "street": "Contact's Street",
        "type": "Contact's Address Type",
        "zip": "Contact's Zip Code"
      }
    ],
    "birthday": "birthday",
    "emails": [
      {
        "email": "email",
        "type": "HOME"
      },
      {
        "email": "email",
        "type": "WORK"
      }
    ],
    "name": {
      "first_name": "first name value",
```

```
    "formatted_name": "formatted name value",
    "last_name": "last name value",
    "suffix": "suffix value"
},
"org": {
    "company": "company name",
    "department": "dep name",
    "title": "title"
},
"phones": [
    {
        "phone": "Phone number",
        "wa-id": "WA-ID value",
        "type": "MAIN"
    },
    {
        "phone": "Phone number",
        "type": "HOME"
    },
    {
        "phone": "Phone number",
        "type": "WORK"
    }
],
"urls": [{
    "url": "some url",
```

```
            "type": "WORK"
        }]
    }
]
```

LOCATION OBJECT

Name	Description
longitude	**Required.** Longitude of the location.
latitude	**Required.** Latitude of the location.
name	**Required.** Name of the location.
address	**Required.** Address of the location.

TEMPLATE OBJECT

Inside template, you can nest the components and the language objects.

Beginning in v2.27.8, a template's namespace must be the namespace associated with the WABA that owns the phone number in the current WhatsApp Business on-prem client. Otherwise, the message will fail to send.

In addition, from v2.41 and onwards, namespace will be an optional field.

Name	Description
name	**Required.** Name of the template.
language *object*	**Required.** Contains a language object. Specifies the language the template may be rendered in. The language object can contain the following fields: policy*string* – **Required.** The language policy the message should follow. **The only supported option is deterministic.** See Language Policy Options. code*string* – **Required.** The code of the language or locale to use. Accepts both language and language_locale formats (e.g., en and en_US). For all codes, see Supported Languages.
components *array of objects*	**Optional.** Array of components objects containing the parameters of the message.
namespace	**Optional.** Only used for On-Premises API. Namespace of the template.

COMPONENTS OBJECT

Inside components, you can nest the parameters object. Additionally, you can set type to button.

Name	Description *(Click the arrow in the left column for supported options.)*

type
string
	Required.
	Describes the component type.
	
	Example of a components object with an array of parameters object nested inside:
	"components": [{
	"type": "body",
	"parameters": [{
	"type": "text",
	"text": "name"
	},
	{
	"type": "text",
	"text": "Hi there"
	}]
	}]

sub_type *string*	**Required when type=button. Not used for the other types.** Type of button to create.
parameters *array of objects*	**Required when type=button.** Array of parameter objects with the content of the message.

Name	Description *(Click the arrow in the left column for supported options.)*
	For components of type=button, see the button parameter object.
index	**Required when type=button. Not used for the other types.** Position index of the button. You can have up to 10 buttons using index values of 0 to 9.

PARAMETER OBJECT

Name	Description

type	**Required.**
	Describes the parameter type.
	Values: text, currency, date_time, image, document, video
	
	Example of a parameter object with a text value:
	{
	"type": "text",
	"text": "Customer"
	}
	
	Example of a parameter object with a media type value of document:
	{
	"type": "document",

Name	Description

	"document":{
	"id": "doc_id",
	"filename": "doc_name"
	}
	}
	This is also known as a Media message template and they only support PDF documents.
	
	For more information about currency and date_time, see the Localizable Parameters section.

BUTTON TYPE

Inside the components object, you can set type to button. These are the button parameters:

Name	Description
sub_type	**Required.** Type of button being created. **Values:** quick_reply, url, copy_code (available from 2.49 and onwards)
index	**Required.** Position index of the button. You can have up to 10 buttons using index values of 0-9.
parameters	**Required.**

Name	Description
	The parameters for the button, which are set at creation time in your Business Manager. Include the following parameters: • type (Required): Indicates the type of parameter for the button. Supported values are payload , text and coupon_code. • payload (Required for quick_reply buttons): Developer-defined payload that will be returned when the button is clicked in addition to the display text on the button. • text (Required for url buttons): Developer provided suffix that will be appended to a previously created dynamic URL button. • coupon_code (Required for copy_code buttons) (available from 2.49 and onwards): Developer provided coupon code which gets copied when the copy code button is clicked.

Example of button type with sub_type quick_reply:

```
{
   "type": "button",
   "sub_type": "quick_reply",
   "index": 0,
   "parameters":
    [{
        "type": "payload",
        "payload": "Yes-Button-Payload"
    }]
}
```

Example of button type with sub_type copy_code

```
{
  "type": "button",
  "sub_type": "copy_code",
  "index": 0,
  "parameters":
  [{
   "type": "coupon_code",
   "coupon_code": "DISCOUNT20"
  }]
}
```

HSM OBJECT

The hsm object has been deprecated with v2.39 of the WhatsApp Business on-premises/reference. Please use the template object instead.

Name	Description
namespace	**Required.** The namespace to be used. Beginning with v2.2.7, if the namespace does not match up to the element_name, the message fails to send.
element_name	**Required.** The element name that indicates which template to use within the namespace. Beginning with v2.2.7, if the element_name does not match up to the namespace, the message fails to send.

Name	Description
language	**Required.** Allows for the specification of a deterministic language. See the Language section for more information. This field used to allow for a fallback option, but this has been deprecated with v2.27.8.
localizable_params	**Required.** This field is an array of values to apply to variables in the template. See the Localizable Parameters section for more information.

INTERACTIVE OBJECT

The interactive object generally contains 4 main components: header, body, footer, and action. Additionally, some of those components can contain one or more different objects:

- Inside header, you can nest media objects.
- Inside action, you can nest section and button objects.

Name	Description
type *string*	**Required.** The type of interactive message you want to send. Supported values: • list: Use it for List Messages. • button: Use it for Reply Buttons.

Name	Description
	• product: Use it for Single-Product Messages. • product_list: Use it for Multi-Product Messages. • catalog_message: Use it for Catalog Messages. • flow: Use it for Flows Messages.
header *object*	**Required for type product_list. Optional for other types.** Header content displayed on top of a message. You cannot set a header if your interactive object is of product type. The header object contains the following fields: document*object* – **Required if type is set to document.** Contains the media object with the document. image*object* – **Required if type is set to image.** Contains the media object with the image. video*object* – **Required if type is set to video.** Contains the media object with the video. text*string* – **Required if type is set to text.** Text for the header. Formatting allows emojis, but not markdown. Maximum length: 60 characters. type*string* – **Required.** The header type you would like to use. Supported values are: text – for List Messages, Reply Buttons, and Multi-Product Messages. video – for Reply Buttons. image – for Reply Buttons.

Name	Description
	document – for Reply Buttons.
body *object*	**Optional for type product. Required for other message types.** An object with the body of the message. The body object contains the following field: text*string* – **Required if body is present.** The content of the message. Emojis and markdown are supported. Maximum length: 1024 characters.
footer *object*	**Optional.** An object with the footer of the message. The footer object contains the following field: text*string* – **Required if footer is present.** The footer content. Emojis, markdown, and links are supported. Maximum length: 60 characters.
action *object*	**Required.** An action object with what you want the user to perform after reading the message. See action object for full information.

Action Object

Name	Description
button *string*	**Required for List Messages.** Button content. It cannot be an empty string and must be unique within the message. Emojis are supported, markdown is not. Maximum length: 20 characters.
buttons *object*	**Required for Reply Button Messages.** A button object. The object can contain the following parameters: type – The only supported option is reply for Reply Button Messages. title – Button title. It cannot be an empty string and must be unique within the message. Emojis are supported, markdown is not. Maximum length: 20 characters. id – Unique identifier for your button. This ID is returned in the webhook when the button is clicked by the user. Maximum length: 256 characters. **You cannot have leading or trailing spaces when setting the ID.**
sections *array of objects*	**Required for List Messages and Multi-Product Messages.** Array of section objects. There is a minimum of 1 and maximum of 10. See section object.
catalog_id *string*	**Required for Single-Product Messages and Multi-Product Messages.**

Name	Description
	Unique identifier of the Facebook catalog linked to your WhatsApp Business Account. This ID can be retrieved via Commerce Manager.
product_retailer_id *string*	**Required for Single-Product Messages and Multi-Product Messages.** Unique identifier of the product in a catalog. **Maximum 100 characters for both Single-Product and Multi-Product messages.** To get this ID, go to Commerce Manager, select your Facebook Business account, and you will see a list of shops connected to your account. Click the shop you want to use. On the left-side panel, click **Catalog > Items**, and find the item you want to mention. The ID for that item is displayed under the item's name.
mode *string*	**Optional for Flows Messages.** The current mode of the Flow, either draft or published. Default: published
flow_message_version *string*	**Required for Flows Messages.** Must be 3.
flow_token *string*	**Required for Flows Messages.** A token that is generated by the business to serve as an identifier.

Name	Description
flow_id *string*	**Required for Flows Messages.** Unique identifier of the Flow provided by WhatsApp.
flow_cta *string*	**Required for Flows Messages.** Text on the CTA button, eg. "Signup". Maximum length: 20 characters (no emoji).
flow_action *string*	**Optional for Flows Messages.** navigate or data_exchange. Use navigate to predefine the first screen as part of the message. Use data_exchange for advanced use-cases where the first screen is provided by your endpoint. Default: navigate
flow_action_payload *object*	**Optional for Flows Messages.** Required only if flow_action is navigate. The object can contain the following parameters: screen*string* – **Required.** The id of the first screen of the Flow. data*object* – **Optional.** The input data for the first screen of the Flow. Must be a non-empty object.

SECTION OBJECT

Name	Description
title *string*	**Required if the message has more than one section.** Title of the section. Maximum length: 24 characters.
rows *array of objects*	**Required for List Messages.** Contains a list of row objects. Limited to 10 rows across all sections. Each row object contains the following fields: title*string* – **Required.** Maximum length: 24 characters. ID*string* – **Required.** Maximum length: 200 characters. description*string* – **Optional.** Maximum length: 72 characters.
product_items *array of objects*	**Required for Multi-Product Messages.** Array of product objects. There is a minimum of 1 product per section and a maximum of 30 products across all sections. Each product object contains the following field: product_retailer_id*string* – **Required for Multi-Product Messages.** Unique identifier of the product in a catalog. To get this ID, go to Commerce Manager, select your account and the shop you want to use. Then, click **Catalog** > **Items**, and find the item you want to mention. The ID for that item is displayed under the item's name.

MESSAGES

/v1/messages

Use the messages node to send text, media, contacts, locations, and interactive messages, as well as message templates to your customers.

See the following guides for information regarding the specific types of messages you can send: Text Messages, Media Messages, Contacts and Location Messages, Interactive Messages, Message Templates, Media Message Templates, and Interactive Message Templates.

EXAMPLES

AUDIO MESSAGES:

```
POST /v1/messages

{

  "recipient_type": "individual",

  "to": "whatsapp-id",

  "type": "audio",

  "audio": {

      "id": "your-media-id",

  }

}
```

DOCUMENT MESSAGES, USING FILENAME:

```
POST /v1/messages

{
```

```
  "recipient_type": "individual",
  "to": "whatsapp-id",
  "type": "document",
  "document": {
      "id": "your-media-id",
      "filename": "your-document-filename"
  }
}
```

DOCUMENT MESSAGES, USING LINK:

```
POST /v1/messages
{
  "recipient_type": "individual",
  "to": "whatsapp-id",
  "type": "document",
  "document": {
      "link": "http(s)://the-url"
      "provider": {
          "name" : "provider-name"
      }
    }
}
```

VIDEO MESSAGES, USING LINK:

```
POST /v1/messages

{

  "recipient_type": "individual",

  "to": "whatsapp-id",

  "type": "video",

  "video": {

      "link": "http(s)://the-url"

      "provider": {

          "name" : "provider-name"

      }

    }

  }

}
```

TEXT MESSAGES:

```
POST /v1/messages

{

  "recipient_type": "individual",

  "to": "whatsapp-id",

  "type": "text",

  "text": {

      "body": "your-message-content"

  }
```

```
}
```

INTERACTIVE MESSAGES (LISTS):

```
POST /v1/messages

{

  "recipient_type": "individual",

  "to": "whatsapp-id",

  "type": "interactive",

  "interactive":{

    "type": "list",

    "header": {

      "type": "text",

      "text": "your-header-content-here"

    },

    "body": {

      "text": "your-text-message-content-here"

    },

    "footer": {

      "text": "your-footer-content-here"

    },

    "action": {

      "button": "cta-button-content-here",

      "sections":[
```

```
      {
        "title":"your-section-title-content-here",
        "rows": [
          {
            "id":"unique-row-identifier-here",
            "title": "row-title-content-here",
            "description": "row-description-content-here",
          }
        ]
      },
      {
        "title":"your-section-title-content-here",
        "rows": [
          {
            "id":"unique-row-identifier-here",
            "title": "row-title-content-here",
            "description": "row-description-content-here",
          }
        ]
      },
      ...
    ]
  }
```

```
  }


}
```

INTERACTIVE MESSAGES (REPLY BUTTONS):

```
POST /v1/messages

{
  "recipient_type": "individual",
  "to": "whatsapp-id",
  "type": "interactive",
  "interactive": {
    "type": "button",
    "header": { # optional
      "type": "text" | "image" | "video" | "document",
      "text": "your text"
      # OR
      "document": {
        "id": "your-media-id",
        "filename": "some-file-name"
      }
      # OR
      "document": {
        "link": "the-provider-name/protocol://the-url",
```

```
    "provider": {
      "name": "provider-name",
    },
    "filename": "some-file-name"
  },
  # OR
  "video": {
    "id": "your-media-id"
  }
  # OR
  "video": {
    "link": "the-provider-name/protocol://the-url",
    "provider": {
      "name": "provider-name"
    }
  }
  # OR
  "image": {
    "id": "your-media-id"
  }
  # OR
  "image": {
    "link": "http(s)://the-url",
```

```
      "provider": {
        "name": "provider-name"
      }
    }
  }, # end header
  "body": {
    "text": "your-text-body-content"
  },
  "footer": { # optional
    "text": "your-text-footer-content"
  },
  "action": {
    "buttons": [
      {
        "type": "reply",
        "reply": {
          "id": "unique-postback-id",
          "title": "First Button's Title"
        }
      },
      {
        "type": "reply",
        "reply": {
```

```
            "id": "unique-postback-id",
            "title": "Second Button’s Title"
          }
        }
      ]
    } # end action
  } # end interactive

}
```

INTERACTIVE MESSAGES (MULTI AND SINGLE-PRODUCT MESSAGES):

```
{
  "recipient_type": "individual",
  "to" : "{{Recipient-WA-ID}}",
  "type": "interactive",
  "interactive": {
    "type": "product",
    "body": {
      "text": "body text"
    },
    "footer": {
      "text": "footer text"
    },
```

```
    "action": {
      "
      _id": "catalog-ID",
      "product_retailer_id": "product-ID"
    }
  }
}
```

INTERACTIVE MESSAGES (MULTI-PRODUCT MESSAGES):

```
{
  "recipient_type": "individual",
  "to" : "whatsapp-id",
  "type": "interactive",
  "interactive":
    {
    "type": "product_list",
    "Header":{
       "type": "text",
        "text": "text-header-content"
     },
     "body":{
        "text": "text-body-content"
```

```
 },
"footer":{
   "text":"text-footer-content"
},
"action":{
   "catalog_id":"catalog-id",
   "sections": [
        {
        "title": "section-title",
        "product_items": [
             { "product_retailer_id": "product-SKU-in-catalog" },
             { "product_retailer_id": "product-SKU-in-catalog" },
                        ...
         ]},
         {
         "title": "the-section-title",
         "product_items": [
            { "product_retailer_id": "product-SKU-in-catalog" }
                       ...
         ]},
          ...
  ]
},
```

```
    }
}
```

INTERACTIVE MESSAGES (CATALOG MESSAGES):

```
{
  "recipient_type": "individual",
  "to" : "whatsapp-id",
  "type": "interactive",
  "interactive":
    {
    "type": "catalog_message",
     "body":{
        "text": "text-body-content"
      },
     "footer":{
        "text":"text-footer-content"
     },
     "action":{
        "name": "catalog_message",
        "parameters":{
          "thumbnail_product_retailer_id": "product-SKU-in-catalog"
        }
     },
```

```
    }
}
```

INTERACTIVE MESSAGES (FLOWS):

```
{
  "recipient_type": "individual",
  "to": "{{Recipient-WA-ID}}",
  "type": "interactive",
  "interactive": {
    "type": "flow",
    "header": {
      "type": "text",
      "text": "Flow message header"
    },
    "body": {
      "text": "Flow message body"
    },
    "footer": {
      "text": "Flow message footer"
    },
    "action": {
      "name": "flow",
      "parameters": {
```

```
        "flow_message_version": "3",
        "flow_token": "AQAAAAACS5FpgQ_cAAAAAD0QI3s",
        "flow_id": "<FLOW_ID>",
        "flow_cta": "Book!",
        "flow_action": "navigate",
        "flow_action_payload": {
          "screen": "<SCREEN_ID>",
          "data": { # optional
            "user_name": "name",
            "user_age": 25
          }
        }
      }
    }
  }
}
```

MEDIA

/v1/media

Use the media node to upload, retrieve, or delete media.

EDGES

The following edges are connected to this node:

Edge	Description
/{media-id}	Use this edge to retrieve and delete media.

BEFORE YOU START

When a media message is sent, the media is stored on the WhatsApp servers for **14 days**. If a user makes a request to download the media after 14 days, the WhatsApp servers will request the same media file from the WhatsApp Business on-premises client. If the media has been removed, the user will be notified that the media is unavailable.

It is not safe to assume the media was downloaded simply based on the delivered and read receipts. Outgoing media is generally safe to be removed past 30 days, but you should employ a strategy that best suits your business.

CONSTRAINTS

- If you use the media upload process rather than linking to a media URL, the file **must** be uploaded to the media volume. Once the upload is complete, you can send a message using the media ID.
- The application processes the media that is uploaded before it's sent to the server. While the maximum file size for media that can be uploaded to the media node is **100MB**, there are post-processing limits for the various media types outlined in the Post-Processing Media Size table below.
- Media storage needs to be handled by the business. If the media volume gets full, message sending will start to fail.
- There is no support for:
    - Sending media by byte streams.
    - Sending messages with animated stickers.

UPLOADING

Make a POST request to /v1/media to upload your media. The body of the on-premises request must contain the binary media data and the Content-Type header must be set to the type of the media being uploaded. See the Supported Content-Types section for supported options.

Sending binary data in a POST HTTP request is a standard way of uploading binary data. If you want to upload an image, for example, you issue a POST request with the actual image bytes in the payload. Alternatively, you can use --data-binary if you want cURL to read and use the given file in binary exactly as given.

EXAMPLE

UPLOADING MEDIA:

```
POST /v1/media

Content-Type: image/jpeg or other appropriate media type

your-binary-media-data
```

UPLOADING MEDIA WITH CURL:

```
curl -X POST \

https://your-webapp-hostname:your-webapp-port/v1/media \

-H 'Authorization: Bearer your-auth-token' \

-H 'Content-Type: image/jpeg'  \ # or other appropriate media type

--data-binary @your-file-path
```

In both cases, a successful response returns the id field, which you need for retrieving media or sending a media message to your customers.

```
{

    "media": [

        {

            "id": "f043afd0-f0ae-4b9c-ab3d-696fb4c8cd68"

        }
```

```
    ]
}
```

If you receive an error message, see Error and Status Messages for more information.

SUPPORTED CONTENT-TYPES

Media	Supported Content-Types
audio	audio/aac, audio/mp4, audio/amr, audio/mpeg, audio/ogg; codecs=opus **Note:** For ogg/opus, only single channel audio files are supported by WA clients.
document	Any valid MIME-type.
image	image/jpeg, image/png Currently, we don't support images with transparent backgrounds.
sticker	image/webp
video	video/mp4, video/3gpp **Notes:** • Only H.264 video codec and AAC audio codec is supported. • We support videos with single audio stream or no audio stream.

POST-PROCESSING MEDIA SIZE

This is the maximum allowed size of the media file after compression and encryption.

Media Type	Size
audio	16 MB
document	100 MB
image	5 MB
sticker	100 KB
video	16 MB

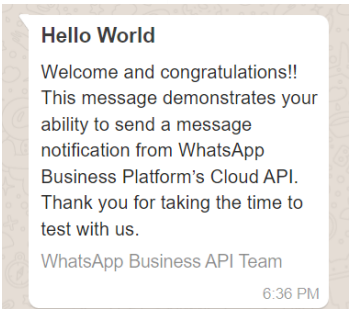
AL CODE EXAMPLES

SEND TEMPLATE AS TEXT

```
procedure SendTemplate()

    var

        Client: HttpClient;
        RequestContent: HttpContent;
        Headers: HttpHeaders;
        Request: HttpRequestMessage;
        Response: HttpResponseMessage;
        ResponseText: Text;
        Jsontext: Text;
        JsonObj1: JsonObject;
        JsonObj2: JsonObject;
        JsonObj3: JsonObject;
        BaseUrl: Text;

    begin
        WhatsappAPISetup.Reset();
        If Not WhatsappAPISetup.Get(1) then
            Error('You must setup ');

        Headers.Clear();
        Clear(BaseUrl);

        //construct Json Object for File Upload

        JsonObj1.Add('messaging_product', 'whatsapp');
        JsonObj1.Add('recipient_type', 'individual');
```

```
        JsonObj1.Add('to', WhatsappAPISetup.PhoneNo);
        //JsonObj1.Add('type', 'text');
        JsonObj1.Add('type', 'template');

        JsonObj3.Add('code', 'en_US');
        JsonObj2.Add('language', JsonObj3);

        JsonObj2.Add('name', 'hello_world');
        JsonObj1.Add('template', JsonObj2);

        JsonObj1.WriteTo(Jsontext);

        Client.DefaultRequestHeaders.Add('Authorization',
WhatsappAPISetup.Authorization);
        RequestContent.WriteFrom(Jsontext);
        Headers := Client.DefaultRequestHeaders();

        RequestContent.GetHeaders(Headers);
        Headers.Remove('Content-Type');
        Headers.Add('Content-Type', 'application/json');


        BaseUrl :=
'https://graph.facebook.com/v17.0/101164682649688/messages';


        if Client.Post(BaseUrl, RequestContent, Response) then begin
            Response.Content().ReadAs(ResponseText);
            if not Response.IsSuccessStatusCode then begin
                Response.Content().ReadAs(ResponseText);
                Message('Something went wrong. \\%1', ResponseText);
            end else begin
                Response.Content().ReadAs(ResponseText);
                Message(ResponseText);
            end;

        end;
    end;
```

OUTPUT



SAMPLE TEXT MESSAGE

```
procedure SendTText()

    var

        Client: HttpClient;
        RequestContent: HttpContent;
        Headers: HttpHeaders;
        Request: HttpRequestMessage;
        Response: HttpResponseMessage;
        ResponseText: Text;
        Jsontext: Text;
        JsonObj1: JsonObject;
        JsonObj2: JsonObject;
        JsonObj3: JsonObject;
        BaseUrl: Text;

    begin

        Headers.Clear();
        Clear(BaseUrl);

        //construct Json Object for File Upload

        JsonObj1.Add('messaging_product', 'whatsapp');
        JsonObj1.Add('recipient_type', 'individual');
        JsonObj1.Add('to', '910987654321');
        JsonObj1.Add('type', 'text');

        JsonObj2.Add('preview_url', 'false');

        JsonObj2.Add('body', 'This msg from Business Central whatsapp
Integration test');
        JsonObj1.Add('text', JsonObj2);

        JsonObj1.WriteTo(Jsontext);
```

```
        Client.DefaultRequestHeaders.Add('Authorization',WhatsappAPISetup.Auth
orization);

        RequestContent.WriteFrom(Jsontext);
        Headers := Client.DefaultRequestHeaders();
        RequestContent.GetHeaders(Headers);
        Headers.Remove('Content-Type');
        Headers.Add('Content-Type', 'application/json');
        BaseUrl :=
'https://graph.facebook.com/v17.0/101144682764688/messages';


        if Client.Post(BaseUrl, RequestContent, Response) then begin
            Response.Content().ReadAs(ResponseText);
            if not Response.IsSuccessStatusCode then begin
                Response.Content().ReadAs(ResponseText);
                Message('Something went wrong. \\%1', ResponseText);
            end else begin
                Response.Content().ReadAs(ResponseText);
                Message(ResponseText);
            end;

        end;
    end;
```

OUTPUT:

This msg from Business Central whatsapp Integration test

SAMPLE SALES INVOICE TEMPLATE WITH PDF ATTACHEMENT

```
  procedure BuildJsonStructureForSalesInv(IdValue: Text; Rec: Record Customer;
FileName: Text[50])
    var
        JsonObj: JsonObject;
        JsonObj1: JsonObject;
        JsonObj2: JsonObject;
        templateObj: JsonObject;
        headerComponent: JsonObject;
        BodyComponent: JsonObject;
        BodyparametersArray: JsonArray;
        HeaderparametersArray: JsonArray;
        documentObj: JsonObject;
        languageObj: JsonObject;
```

```
        componentsArray: JsonArray;
        Client: HttpClient;
        RequestContent: HttpContent;
        Headers: HttpHeaders;
        Request: HttpRequestMessage;
        Response: HttpResponseMessage;
        ResponseText: Text;
        BaseUrl: Text;
        Jsontext: Text;
        WhatsappAPISetup: Record "WhatsApp Integration Setup";
        FinalFileName: Text;
        WhatsAppLogEntry: Record "Whatsapp Log Entry";
    begin

        WhatsappAPISetup.Reset();
        WhatsappAPISetup.SetFilter("Authendication Token", '<>%1', '');
        WhatsappAPISetup.SetFilter("Base API URl", '<>%1', '');
        WhatsappAPISetup.SetFilter("Get Media API URl ", '<>%1', '');
        if Not WhatsappAPISetup.FindFirst() then
            Error('You must configure the WhatsApp Integration Setup');

        if IdValue = '' then
            exit;
        Headers.Clear();
        Clear(BaseUrl);
        Clear(FinalFileName);

        FinalFileName := FileName + '.pdf';

        // Create the main JSON object
        JsonObj.Add('messaging_product', 'whatsapp');
        JsonObj.Add('recipient_type', 'individual');
        JsonObj.Add('to', Rec."Whatsapp CountryCode" + Rec."Whatsapp number");
        JsonObj.Add('type', 'template');

        // Create a JSON object for the 'template' property
        templateObj.Add('name', 'sales_invoice_1');

        // Create a JSON object for the 'language' property


        languageObj.Add('code', 'en_US');
        templateObj.Add('language', languageObj);

        // Create an array for 'components'

        // Create a JSON object for the 'header' component
        headerComponent.Add('type', 'header');
```

```
        // Create an array for 'parameters' in the 'header' component
        HeaderparametersArray.Add(CreateDocumentParameter(IdValue,
FinalFileName));

        // Add the 'parameters' array to the 'header' component
        headerComponent.Add('parameters', HeaderparametersArray);

        // Add the 'header' component to the 'components' array
        componentsArray.Add(headerComponent);

        //*************************************************************
************



        // Create a JSON object for the 'Body' component
        BodyComponent.Add('type', 'body');

        // Create an array for 'parameters' in the 'body' component

        BodyparametersArray.Add(CreateTextParameter(FinalFileName));
        BodyparametersArray.Add(CreateTextParameter('now'));

        // Add the 'parameters' array to the 'body' component
        BodyComponent.Add('parameters', BodyparametersArray);

        // Add the 'body' component to the 'components' array
        componentsArray.Add(BodyComponent);


        // Add the 'components' array to the 'template' object
        templateObj.Add('components', componentsArray);

        // Add the 'template' object to the main JSON object
        JsonObj.Add('template', templateObj);


        // Convert the JSON object to text

        JsonObj.WriteTo(Jsontext);

        //Message(Jsontext);


        Client.DefaultRequestHeaders.Add('Authorization',
WhatsappAPISetup."Authendication Token");
        RequestContent.WriteFrom(Jsontext);
        Headers := Client.DefaultRequestHeaders();

        RequestContent.GetHeaders(Headers);
```

```
        Headers.Remove('Content-Type');
        Headers.Add('Content-Type', 'application/json');


        BaseUrl := WhatsappAPISetup."Base API URl";


        if Client.Post(BaseUrl, RequestContent, Response) then begin
            Response.Content().ReadAs(ResponseText);
            if not Response.IsSuccessStatusCode then begin
                Message('File Send Faild');
            end else begin
                Response.Content().ReadAs(ResponseText);
                Message('File Send Successfully');
            end;

        end;
    end;

    local procedure CreateDocumentParameter(DocumentId: Text; FileName: Text):
JsonObject
    var
        documentObj: JsonObject;
        documentIdObj: JsonObject;
    begin
        // Create a JSON object for a 'document' parameter
        documentObj.Add('type', 'document');

        // Create a JSON object for the 'document' property
        documentIdObj.Add('id', DocumentId);
        documentIdObj.Add('filename', FileName);
        documentObj.Add('document', documentIdObj);

        exit(documentObj);
    end;

    local procedure CreateTextParameter(TextValue: Text): JsonObject
    var
        textObj: JsonObject;
    begin
        // Create a JSON object for a 'text' parameter
        textObj.Add('type', 'text');
        textObj.Add('text', TextValue);

        exit(textObj);
    end;

    procedure SendAttachment(IdValue: Text; Rec: Record Customer; FileName:
Text[50])
```

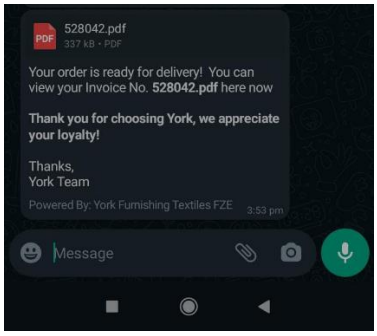
```
    var

        Client: HttpClient;
        RequestContent: HttpContent;
        Headers: HttpHeaders;
        Request: HttpRequestMessage;
        Response: HttpResponseMessage;
        WhatsappAPISetup: Record "WhatsApp Integration Setup";
        ResponseText: Text;
        Jsontext: Text;
        JsonObj1: JsonObject;
        JsonObj2: JsonObject;
        JsonObj3: JsonObject;
        BaseUrl: Text;
        FinalFileName: Text;

    begin

        WhatsappAPISetup.Reset();
        WhatsappAPISetup.SetFilter("Authendication Token", '<>%1', '');
        WhatsappAPISetup.SetFilter("Base API URl", '<>%1', '');
        WhatsappAPISetup.SetFilter("Get Media API URl ", '<>%1', '');
        if Not WhatsappAPISetup.FindFirst() then
            Error('You must configure the WhatsApp Integration Setup');

        if IdValue = '' then
            exit;
        Headers.Clear();
        Clear(BaseUrl);
        Clear(FinalFileName);

        //construct Json Object for File Upload

        JsonObj1.Add('messaging_product', 'whatsapp');
        JsonObj1.Add('recipient_type', 'individual');
        JsonObj1.Add('to', Rec."Whatsapp CountryCode" + Rec."Whatsapp
number");
        JsonObj1.Add('type', 'document');

        JsonObj2.Add('id', IdValue);

        FinalFileName := FileName + '.pdf';
        JsonObj2.Add('filename', FinalFileName);
        JsonObj1.Add('document', JsonObj2);

        JsonObj1.WriteTo(Jsontext);
```

```
        Client.DefaultRequestHeaders.Add('Authorization',
WhatsappAPISetup."Authendication Token");
        RequestContent.WriteFrom(Jsontext);
        Headers := Client.DefaultRequestHeaders();

        RequestContent.GetHeaders(Headers);
        Headers.Remove('Content-Type');
        Headers.Add('Content-Type', 'application/json');


        BaseUrl := WhatsappAPISetup."Base API URl";


        if Client.Post(BaseUrl, RequestContent, Response) then begin
            Response.Content().ReadAs(ResponseText);
            if not Response.IsSuccessStatusCode then begin
                Response.Content().ReadAs(ResponseText);
                Message('File Send Faild');
            end else begin
                Response.Content().ReadAs(ResponseText);
                Message('File Send Successfully');
            end;

        end;
    end;




      procedure GetMediaId(FileInStream: InStream; Rec: Record Customer;
FileName: Text[50]; Format: Integer)
    var
        bfS: TextBuilder;
        afS: TextBuilder;
        TempBlob: Codeunit "Temp Blob";
        PayloadOutStream: OutStream;
        PayloadInStream: InStream;
        Content: HttpContent;
        Headers: HttpHeaders;
        Client: HttpClient;
        Request: HttpRequestMessage;
        Response: HttpResponseMessage;
        ResponseText: Text;
        IdValue: Text;
        WhatsappAPISetup: Record "WhatsApp Integration Setup";
        JObject: JsonObject;
        JToken: JsonToken;
        Token: JsonToken;
        WhatsAppLogEntry: Record "Whatsapp Log Entry";
```

```
    begin

        WhatsappAPISetup.Reset();
        WhatsappAPISetup.SetFilter("Authendication Token", '<>%1', '');
        WhatsappAPISetup.SetFilter("Base API URl", '<>%1', '');
        WhatsappAPISetup.SetFilter("Get Media API URl ", '<>%1', '');
        if Not WhatsappAPISetup.FindFirst() then
            Error('You must configure the WhatsApp Integration Setup');

        bfS.AppendLine('--123456789');
        bfS.AppendLine('Content-Disposition: form-data; name="file";
filename="data.pdf"');
        bfS.AppendLine('Content-Type: application/pdf');
        bfS.AppendLine();
        afS.AppendLine();
        afS.AppendLine('(data)');
        afS.AppendLine('--123456789');
        afS.AppendLine('Content-Disposition: form-data;
name="messaging_product"');
        afS.AppendLine();
        afS.AppendLine('whatsapp');
        afS.AppendLine('--123456789--');
        Clear(TempBlob);
        TempBlob.CreateOutStream(PayloadOutStream);
        PayloadOutStream.WriteText(bfS.ToText());
        CopyStream(PayloadOutStream, FileInStream);
        PayloadOutStream.WriteText(afS.ToText());
        TempBlob.CreateInStream(PayloadInStream);


        //
Request.SetRequestUri('https://graph.facebook.com/v17.0/101144682649688/media'
);
        Request.SetRequestUri(WhatsappAPISetup."Get Media API URl ");
        Request.Method := 'POST';
        Request.GetHeaders(Headers);
        Headers.Add('Authorization', WhatsappAPISetup."Authendication Token");
        Content.WriteFrom(PayloadInStream);
        //Content.WriteFrom(tb.ToText());
        Content.GetHeaders(Headers);
        if headers.Contains('Content-Type') then
            headers.Remove('Content-Type');
        headers.Add('Content-Type', 'multipart/form-data;
boundary="123456789"');
        Request.Content := Content;

        Client.Send(Request, Response);
```

```
        if not Response.IsSuccessStatusCode() then begin
            Response.Content.ReadAs(REsponseText);
            WhatsAppLogEntry.Init();
            If Format = 1 then
                WhatsAppLogEntry."Template Name" := 'Sales Invoice - ' +
FileName
            else
            Message('File Send Failed');

        end else begin
            Response.Content.ReadAs(ResponseText);

            JObject.ReadFrom(REsponseText);
            JObject.Get('id', JToken);
            IdValue := DelChr(REsponseText, '=', 'id{}":');
            // SendAttachment(IdValue, Rec, FileName);
            If Format = 1 then
                BuildJsonStructureForSalesInv(IdValue, Rec, FileName)
            else
                If Format = 2 then
                    BuildJsonStructureForSOA(IdValue, Rec, FileName)
        end;
    end;
```

OUTPUT



REFERENCES

META REFERCENCES

1. Meta Developer Documentation | Meta APIs, SDKs & Guides (facebook.com)

2. WhatsApp Business Platform (facebook.com)
3. Phone Numbers - WhatsApp Business Platform (facebook.com)
4. Cloud API (facebook.com)
5. Register - Meta App Development (facebook.com)
6. Create an App - Meta App Development (facebook.com)
7. Pricing - WhatsApp Business Platform (facebook.com)
8. Templates - WhatsApp Business Management API (facebook.com)
9. Message Template Guidelines - WhatsApp Business Platform (facebook.com)
10. Media - Cloud API (facebook.com)
11. Cloud API | WhatsApp Business Platform | Postman API Network
12. https://www.postman.com/meta/workspace/whatsapp-business-platform/collection/13382743-2fd9b32d-f63c-4056-873e-4c398dde9d6d

OTHER REFERENCES

1. Send Message API | WbizTool Integration Documentation
2. WbizTool Integration API Documentation
3. Top 10 Whatsapp API Providers for Business in 2023 (startuptalky.com)
4. Integrate WhatsApp Business with Microsoft Dynamics 365 Business Central, WhatsApp Business Microsoft Dynamics 365 Business Central integration with AI (appypie.com)
5. Configure a WhatsApp channel through Twilio | Microsoft Learn
6. WhatsApp messaging with Business Central (picazin.dev)
7. Integrate the WhatsApp Business API with the Microsoft Dynamics 365 Business Central API API - Pipedream

Prepared by

Janaki P (CIT245)

ERP Technical Consultant

Documented Date: 01- Nav- 2023