Client Reference

Contents

Back to top

- `WhatsApp.update_flow_metadata()`
- `WhatsApp.update_flow_json()`
- `WhatsApp.publish_flow()`
- `WhatsApp.delete_flow()`
- `WhatsApp.deprecate_flow()`
- `WhatsApp.get_flow()`
- `WhatsApp.get_flows()`
- `WhatsApp.get_flow_metrics()`
- `WhatsApp.get_flow_assets()`
- `WhatsApp.register_phone_number()`
- `WhatsApp.create_qr_code()`
- `WhatsApp.get_qr_code()`
- `WhatsApp.get_qr_codes()`
- `WhatsApp.update_qr_code()`
- `WhatsApp.delete_qr_code()`
- `WhatsApp.webhook_update_handler()`
- `WhatsApp.webhook_challenge_handler()`
- `WhatsApp.get_flow_request_handler()`

Hide Search Matches

WhatsApp.send_message(*to: str | int, text: str, header: str | None = None, footer: str | None = None, buttons: Iterable[Button] | ButtonUrl | SectionList | FlowButton | None = None, preview_url: bool = False, reply_to_message_id: str | None = None, keyboard: None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a message to a WhatsApp user.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_message(
...     to="1234567890",
...     text="Hello from PyWa! (https://github.com/david-lev/pywa)",
...     preview_url=True,
... )
```

Example with keyboard buttons:

```
>>> from pywa.types import Button
>>> wa = WhatsApp(...)
>>> wa.send_message(
...     to="1234567890",
...     text="What can I help you with?",
...     footer="Powered by PyWa",
...     buttons=[
...         Button("Help", data="help"),
...         Button("About", data="about"),
...     ],
... )
```

Example with a button url:

```
>>> from pywa.types import ButtonUrl
>>> wa = WhatsApp(...)
>>> wa.send_message(
...     to="1234567890",
...     text="Hello from PyWa!",
...     footer="Powered by PyWa",
...     buttons=ButtonUrl(
...         title="PyWa GitHub",
...         url="https://github.com/david-lev/pywa",
...     )
... )
```

Example with a section list:

```
>>> from pywa.types import SectionList, Section, SectionRow
>>> wa = WhatsApp(...)
>>> wa.send_message(
...     to="1234567890",
...     text="What can I help you with?",
...     footer="Powered by PyWa",
...     buttons=SectionList(
...         button_title="Choose an option",
...         sections=[
...             Section(
...                 title="Help",
...                 rows=[
...                     SectionRow(
...                         title="Help",
...                         callback_data="help",
...                         description="Get help with PyWa",
...                     ),
...                     SectionRow(
...                         title="About",
...                         callback_data="about",
...                         description="Learn more about PyWa",
...                     ),
...                 ],
...             ),
...            Section(
...                 title="Other",
...                 rows=[
...                     SectionRow(
...                         title="GitHub",
...                         callback_data="github",
...                         description="View the PyWa GitHub repository",
...                     ),
...                 ],
...             ),
...         ],
...     ),
... )
```

Example with a flow button:

```
>>> from pywa.types import FlowButton, FlowActionType
>>> wa = WhatsApp(...)
>>> wa.send_message(
...     to="1234567890",
...     text="We love to get feedback!",
...     footer="Powered by PyWa",
...     buttons=FlowButton(
...         title="Feedback",
...         flow_id="1234567890",
...         flow_token="AQAAAAACS5FpgQ_cAAAAAD0QI3s.",
...         flow_action_type=FlowActionType.NAVIGATE,
...         flow_action_screen="RECOMMENDED"
...     ),
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **text** – The text to send (markdown allowed, max 4096 characters).
- **header** –

  The header of the message (if `buttons` are provided, optional, up to 60 characters, no markdown allowed).
- **footer** –

  The footer of the message (if `buttons` are provided, optional, up to 60 characters, markdown has no effect).
- **buttons** – The buttons to send with the message (optional).
- **preview_url** – Whether to show a preview of the URL in the message (if any).
- **reply_to_message_id** – The message ID to reply to (optional).
- **keyboard** – Deprecated and will be removed in a future version, use `buttons` instead.
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent message.

```
WhatsApp.send_image(to: str | int, image: str | Path | bytes | BinaryIO,
caption: str | None = None, body: None = None, footer: str | None = None,
buttons: Iterable[Button] | ButtonUrl | FlowButton | None = None,
reply_to_message_id: str | None = None, mime_type: str | None = None,
tracker: CallbackDataT | None = None, sender: str | int | None = None) →
str
```

Send an image to a WhatsApp user.

- Images must be 8-bit, RGB or RGBA.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_image(
...     to="1234567890",
...     image="https://example.com/image.png",
...     caption="This is an image!",
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **image** – The image to send (either a media ID, URL, file path, bytes, or an open file object. When buttons are provided, only URL is supported).
- **caption** –

  The caption of the image (required when buttons are provided, markdown allowed).
- **footer** –

  The footer of the message (if buttons are provided, optional, markdown has no effect).
- **buttons** – The buttons to send with the image (optional).
- **reply_to_message_id** – The message ID to reply to (optional, only works if buttons provided).
- **mime_type** – The mime type of the image (optional, required when sending an image as bytes or a file object, or file path that does not have an extension).
- **body** – Deprecated and will be removed in a future version, use `caption` instead.
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent image message.

WhatsApp.send_video(*to: str | int, video: str | Path | bytes | BinaryIO, caption: str | None = None, body: None = None, footer: str | None = None, buttons: Iterable[Button] | ButtonUrl | FlowButton | None = None, reply_to_message_id: str | None = None, mime_type: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a video to a WhatsApp user.

- Only H.264 video codec and AAC audio codec is supported.
- Videos with a single audio stream or no audio stream are supported.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_video(
...     to="1234567890",
...     video="https://example.com/video.mp4",
...     caption="This is a video",
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **video** – The video to send (either a media ID, URL, file path, bytes, or an open file object. When buttons are provided, only URL is supported).
- **caption** –

  The caption of the video (required when sending a video with buttons, markdown allowed).
- **footer** –

  The footer of the message (if buttons are provided, optional, markdown has no effect).
- **buttons** – The buttons to send with the video (optional).
- **reply_to_message_id** – The message ID to reply to (optional, only works if buttons provided).
- **mime_type** – The mime type of the video (optional, required when sending a video as bytes or a file object, or file path that does not have an extension).
- **body** – Deprecated and will be removed in a future version, use `caption` instead.
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent video.

WhatsApp.send_audio(*to: str | int, audio: str | Path | bytes | BinaryIO, mime_type: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send an audio file to a WhatsApp user.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_audio(
...     to='1234567890',
...     audio='https://example.com/audio.mp3',
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **audio** – The audio file to send (either a media ID, URL, file path, bytes, or an open file object).
- **mime_type** – The mime type of the audio file (optional, required when sending an audio file as bytes or a file object, or file path that does not have an extension).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent audio file.

WhatsApp.send_document(*to: str | int, document: str | Path | bytes | BinaryIO, filename: str | None = None, caption: str | None = None, body: None = None, footer: str | None = None, buttons: Iterable[Button] | ButtonUrl | FlowButton | None = None, reply_to_message_id: str | None = None, mime_type: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a document to a WhatsApp user.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_document(
...     to="1234567890",
...     document="https://example.com/example_123.pdf",
...     filename="example.pdf",
...     caption="Example PDF"
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **document** – The document to send (either a media ID, URL, file path, bytes, or an open file object. When buttons are provided, only URL is supported).
- **filename** – The filename of the document (optional, The extension of the filename will specify what format the document is displayed as in WhatsApp).
- **caption** –

  The caption of the document (required when sending a document with buttons, markdown allowed).
- **footer** –

  The footer of the message (if buttons are provided, optional, markdown has no effect).
- **buttons** – The buttons to send with the document (optional).
- **reply_to_message_id** – The message ID to reply to (optional, only works if buttons provided).
- **mime_type** – The mime type of the document (optional, required when sending a document as bytes or a file object, or file path that does not have an extension).
- **body** – Deprecated and will be removed in a future version, use `caption` instead.
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent document.

WhatsApp.send_location(*to: str | int, latitude: float, longitude: float, name: str | None = None, address: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a location to a WhatsApp user.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_location(
...     to='1234567890',
...     latitude=37.4847483695049,
...     longitude=--122.1473373086664,
...     name='WhatsApp HQ',
...     address='Menlo Park, 1601 Willow Rd, United States',
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **latitude** – The latitude of the location.
- **longitude** – The longitude of the location.
- **name** – The name of the location (optional).
- **address** – The address of the location (optional).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent location.

WhatsApp.send_contact(*to: str | int, contact: Contact | Iterable[Contact], reply_to_message_id: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a contact/s to a WhatsApp user.

Example

```
>>> from pywa.types import Contact
>>> wa = WhatsApp(...)
>>> wa.send_contact(
...     to='1234567890',
...     contact=Contact(
...         name=Contact.Name(formatted_name='David Lev', first_name='David'),
...         phones=[Contact.Phone(phone='1234567890', wa_id='1234567890', type=
...         emails=[Contact.Email(email='test@test.com', type='WORK')],
...         urls=[Contact.Url(url='https://exmaple.com', type='HOME')],
...      )
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **contact** – The contact/s to send.
- **reply_to_message_id** – The message ID to reply to (optional).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent message.

WhatsApp.send_sticker(*to: str | int, sticker: str | Path | bytes | BinaryIO, mime_type: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a sticker to a WhatsApp user.

- A static sticker needs to be 512x512 pixels and cannot exceed 100 KB.
- An animated sticker must be 512x512 pixels and cannot exceed 500 KB.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_sticker(
...     to='1234567890',
...     sticker='https://example.com/sticker.webp',
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **sticker** – The sticker to send (either a media ID, URL, file path, bytes, or an open file object).
- **mime_type** – The mime type of the sticker (optional, required when sending a sticker as bytes or a file object, or file path that does not have an extension).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent message.

`WhatsApp.send_catalog(to: str | int, body: str, footer: str | None = None, thumbnail_product_sku: str | None = None, reply_to_message_id: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None) → str`

Send the business catalog to a WhatsApp user.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_catalog(
...     to='1234567890',
...     body='Check out our catalog!',
...     footer='Powered by PyWa',
...     thumbnail_product_sku='SKU123',
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **body** – Text to appear in the message body (up to 1024 characters).
- **footer** – Text to appear in the footer of the message (optional, up to 60 characters).
- **thumbnail_product_sku** – The thumbnail of this item will be used as the message's header image (optional, if not provided, the first item in the catalog will be used).
- **reply_to_message_id** – The message ID to reply to (optional).

- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent message.

WhatsApp.send_template(*to: str | int, template: Template, reply_to_message_id: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Send a template to a WhatsApp user.

- To create a template, use `create_template()`.

Example

```
>>> from pywa.types import Template as Temp
>>> wa = WhatsApp(...)
>>> wa.send_template(
...     to='1234567890',
...         template=Temp(
...         name='buy_new_iphone_x',
...         language=Temp.Language.ENGLISH_US,
...         header=Temp.TextValue(value='15'),
...         body=[
...             Temp.TextValue(value='John Doe'),
...             Temp.TextValue(value='WA_IPHONE_15'),
...             Temp.TextValue(value='15%'),
...         ],
...         buttons=[
...             Temp.UrlButtonValue(value='iphone15'),
...             Temp.QuickReplyButtonData(data='unsubscribe_from_marketing_mess
...             Temp.QuickReplyButtonData(data='unsubscribe_from_all_messages')
...         ],
...     ),
... )
```

Example for Authentication Template:

```
>>> from pywa.types import Template as Temp
>>> wa = WhatsApp(...)
>>> wa.send_template(
...     to='1234567890',
...         template=Temp(
...         name='auth_with_otp',
...         language=Temp.Language.ENGLISH_US,
...         buttons=Temp.OTPButtonCode(code='123456'),
...     ),
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **template** – The template to send.
- **reply_to_message_id** – The message ID to reply to (optional).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent template.

Raises:

***WhatsApp.*send_product**(*to: str | int, catalog_id: str, sku: str, body: str | None = None, footer: str | None = None, reply_to_message_id: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str**

Send a product from a business catalog to a WhatsApp user.

- To send multiple products, use `send_products()`.

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_product(
...     to='1234567890',
...     catalog_id='1234567890',
...     sku='SKU123',
...     body='Check out this product!',
...     footer='Powered by PyWa',
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **catalog_id** – The ID of the catalog to send the product from. (To get the catalog ID use `get_commerce_settings()` or in the Commerce Manager).
- **sku** – The product SKU to send.
- **body** – Text to appear in the message body (up to 1024 characters).
- **footer** – Text to appear in the footer of the message (optional, up to 60 characters).
- **reply_to_message_id** – The message ID to reply to (optional).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent message.

`WhatsApp.send_products(to: str | int, catalog_id: str, product_sections: Iterable[ProductsSection], title: str, body: str, footer: str | None = None, reply_to_message_id: str | None = None, tracker: CallbackDataT | None = None, sender: str | int | None = None) → str`

Send products from a business catalog to a WhatsApp user.

- To send a single product, use `send_product()`.

Example

```
>>> from pywa.types import ProductsSection
>>> wa = WhatsApp(...)
>>> wa.send_products(
...     to='1234567890',
...     catalog_id='1234567890',
...     title='Tech Products',
...     body='Check out our products!',
...     product_sections=[
...         ProductsSection(
...             title='Smartphones',
...             skus=['IPHONE12', 'GALAXYS21'],
...         ),
...         ProductsSection(
...             title='Laptops',
...             skus=['MACBOOKPRO', 'SURFACEPRO'],
...         ),
...     ],
...     footer='Powered by PyWa',
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **catalog_id** –

  The ID of the catalog to send the product from (To get the catalog ID use `get_commerce_settings()` or in the Commerce Manager).
- **product_sections** – The product sections to send (up to 30 products across all sections).
- **title** – The title of the product list (up to 60 characters).
- **body** – Text to appear in the message body (up to 1024 characters).
- **footer** – Text to appear in the footer of the message (optional, up to 60 characters).
- **reply_to_message_id** – The message ID to reply to (optional).
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the sent message.

WhatsApp.send_reaction(*to: str | int, emoji: str, message_id: str, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

React to a message with an emoji.

- You can react to incoming messages by using the `react()` method on every update.

```
>>> wa = WhatsApp(...)
>>> @wa.on_message()
... def message_handler(_: WhatsApp, msg: Message):
...     msg.react('👍')
```

Example

```
>>> wa = WhatsApp(...)
>>> wa.send_reaction(
...     to='1234567890',
...     emoji='👍',
...     message_id='wamid.XXX='
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **emoji** – The emoji to react with.
- **message_id** – The message ID to react to.
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the reaction (You can't use this message id to remove the reaction or perform any other action on it. instead, use the message ID of the message you reacted to).

WhatsApp.remove_reaction(*to: str | int, message_id: str, tracker: CallbackDataT | None = None, sender: str | int | None = None*) → str

Remove reaction from a message.

- You can remove reactions from incoming messages by using the `unreact()` method on every update.

```
>>> wa = WhatsApp(...)
>>> @wa.on_message()
... def message_handler(_: WhatsApp, msg: Message):
...     msg.unreact()
```

Example

```
>>> wa = WhatsApp(...)
>>> wa.remove_reaction(
...     to='1234567890',
...     message_id='wamid.XXX='
... )
```

Parameters:

- **to** – The phone ID of the WhatsApp user.
- **message_id** – The message ID to remove the reaction from.
- **tracker** – The data to track the message with (optional, up to 512 characters, for complex data You can use `CallbackData`).
- **sender** – The phone ID to send the message from (optional, overrides the client's phone ID).

Returns:

The message ID of the reaction (You can't use this message id to re-react or perform any other action on it. instead, use the message ID of the message you unreacted to).

WhatsApp.mark_message_as_read(*message_id: str, sender: str | int | None = None*) → bool

Mark a message as read.

- You can mark incoming messages as read by using the `mark_as_read()` method.

Example

```
>>> wa = WhatsApp(...)
>>> wa.mark_message_as_read(message_id='wamid.XXX=')
```

Parameters:

- **message_id** – The message ID to mark as read.

- **sender** – The phone ID (optional, if not provided, the client's phone ID will be used).

Returns:

Whether the message was marked as read.

WhatsApp.upload_media(*media: str | Path | bytes | BinaryIO, mime_type: str | None = None, filename: str | None = None, dl_session: Client | None = None, phone_id: str | None = None*) → str

Upload media to WhatsApp servers.

Example

```
>>> wa = WhatsApp(...)
>>> wa.upload_media(
...     media='https://example.com/image.jpg',
...     mime_type='image/jpeg',
... )
```

Parameters:

- **media** – The media to upload (can be a URL, bytes, or a file path).
- **mime_type** – The MIME type of the media (required if media is bytes or a file path).
- **filename** – The file name of the media (required if media is bytes).
- **dl_session** – A httpx client to use when downloading the media from a URL (optional, if not provided, a new session will be created).
- **phone_id** – The phone ID to upload the media to (optional, if not provided, the client's phone ID will be used).

Returns:

The media ID.

Raises:

ValueError –

- If provided `media` is file path and the file does not exist. - If provided `media` is URL and the URL is invalid or media cannot be downloaded. - If provided `media` is bytes and `filename` or `mime_type` is not provided.

`WhatsApp.download_media(url: str, path: str | None = None, filename: str | None = None, in_memory: bool = False, **kwargs) → str | bytes`

Download a media file from WhatsApp servers.

Example

```
>>> wa = WhatsApp(...)
>>> wa.download_media(
...     url='https://mmg-fna.whatsapp.net/d/f/Amc.../v2/1234567890',
...     path='/home/david/Downloads',
...     filename='image.jpg',
... )
```

Parameters:

- **url** – The URL of the media file (from `get_media_url()`).
- **path** – The path where to save the file (if not provided, the current working directory will be used).
- **filename** – The name of the file (if not provided, it will be guessed from the URL + extension).
- **in_memory** – Whether to return the file as bytes instead of saving it to disk (default: False).
- ****kwargs** – Additional arguments to pass to `httpx.get()`.

Returns:

The path of the saved file if `in_memory` is False, the file as bytes otherwise.

`WhatsApp.get_media_url(media_id: str) → MediaUrlResponse`

Get the URL of a media.

- The URL is valid for 5 minutes.
- The media can be downloaded directly from the message using the `download_media()` method.

Example

```
>>> wa = WhatsApp(...)
>>> wa.get_media_url(media_id='wamid.XXX=')
```

Parameters:

media_id – The media ID.

Returns:

A MediaResponse object with the media URL.

WhatsApp.get_business_profile(*phone_id: str | None = None*) → BusinessProfile

Get the business profile of the WhatsApp Business account.

Example

```
>>> wa = WhatsApp(...)
>>> wa.get_business_profile()
```

Parameters:

phone_id – The phone ID to get the business profile from (optional, if not provided, the client's phone ID will be used).

Returns:

The business profile.

WhatsApp.get_business_phone_number(*phone_id: str | None = None*) → BusinessPhoneNumber

Get the phone number of the WhatsApp Business account.

Example

```
>>> wa = WhatsApp(...)
>>> wa.get_business_phone_number()
```

Parameters:

phone_id – The phone ID to get the phone number from (optional, if not provided, the client's phone ID will be used).

Returns:

The phone number object.

WhatsApp.update_business_profile(*about: str | None = <object object>, address: str | None = <object object>, description: str | None = <object object>, email: str | None = <object object>, profile_picture_handle: str | None = <object object>, industry: ~pywa.types.others.Industry | None = <object object>, websites: ~typing.Iterable[str] | None = <object object>, phone_id: str | None = None*) → bool

Update the business profile of the WhatsApp Business account.

Example

```
>>> from pywa.types import Industry
>>> wa = WhatsApp(...)
>>> wa.update_business_profile(
...     about='This is a test business',
...     address='Menlo Park, 1601 Willow Rd, United States',
...     description='This is a test business',
...     email='test@test.com',
...     profile_picture_handle='1234567890',
...     industry=Industry.NOT_A_BIZ,
...     websites=('https://example.com', 'https://google.com'),
... )
```

Parameters:

- **about** –

  The business's About text. This text appears in the business's profile, beneath its profile image, phone number, and contact buttons. (cannot be empty. must be between 1 and 139 characters. markdown is not supported. Hyperlinks can be included but will not render as clickable links.)
- **address** – Address of the business. Character limit 256.
- **description** – Description of the business. Character limit 512.
- **email** – The contact email address (in valid email format) of the business. Character limit 128.
- **profile_picture_handle** – Handle of the profile picture. This handle is generated when you upload the binary file for the profile picture to Meta using the Resumable Upload API.
- **industry** – Industry of the business.
- **websites** – The URLs associated with the business. For instance, a website, Facebook Page, or Instagram. (You must include the `http://` or `https://` portion of the URL. There is a maximum of 2 websites with a maximum of 256 characters each.)

- **phone_id** – The phone ID to update the business profile for (optional, if not provided, the client's phone ID will be used).

Returns:

Whether the business profile was updated.

`WhatsApp.update_conversational_automation(enable_chat_opened: bool, ice_breakers: Iterable[str] | None = None, commands: Iterable[Command] | None = None, phone_id: str | None = None) → bool`

Update the conversational automation settings of the WhatsApp Business account.

- You can receive the current conversational automation settings using `get_business_phone_number()` and accessing the `conversational_automation` attribute.
- Read more about Conversational Automation.

Parameters:

- **enable_chat_opened** – You can be notified whenever a WhatsApp user opens a chat with you for the first time. This can be useful if you want to reply to these users with a special welcome message of your own design (When enabled, you'll start receiving the `ChatOpened` event).
- **ice_breakers** – Ice Breakers are customizable, tappable text strings that appear in a message thread the first time you chat with a user. For example, *Plan a trip* or *Create a workout plan*.
- **commands** – Commands are text strings that WhatsApp users can see by typing a forward slash in a message thread with your business.
- **phone_id** – The phone ID to update the conversational automation settings for (optional, if not provided, the client's phone ID will be used).

Returns:

Whether the conversational automation settings were updated.

`WhatsApp.set_business_public_key(public_key: str, phone_id: str | None = None) → bool`

Set the business public key of the WhatsApp Business account (required for end-to-end encryption in flows)

Parameters:

- **public_key** – An public 2048-bit RSA Key in PEM format.
- **phone_id** – The phone ID to set the business public key for (optional, if not provided, the client's phone ID will be used).

Example

```
>>> wa = WhatsApp(...)
>>> wa.set_business_public_key(
...     public_key="""-----BEGIN PUBLIC KEY-----..."""
... )
```

Returns:

Whether the business public key was set.

WhatsApp.get_commerce_settings(*phone_id: str | None = None*) → CommerceSettings

Get the commerce settings of the WhatsApp Business account.

Example

```
>>> wa = WhatsApp(...)
>>> wa.get_commerce_settings()
```

Returns:

The commerce settings.

WhatsApp.update_commerce_settings(*is_catalog_visible: bool = None*, *is_cart_enabled: bool = None*, *phone_id: str | None = None*) → bool

Update the commerce settings of the WhatsApp Business account.

Example

```
>>> wa = WhatsApp(...)
>>> wa.update_commerce_settings(
...     is_catalog_visible=True,
...     is_cart_enabled=True,
... )
```

Parameters:

- **is_catalog_visible** – Whether the catalog is visible (optional).
- **is_cart_enabled** – Whether the cart is enabled (optional).
- **phone_id** – The phone ID to update the commerce settings for (optional, if not provided, the client's phone ID will be used).

Returns:

Whether the commerce settings were updated.

Raises:

ValueError – If no arguments are provided.

`WhatsApp.create_template(template: NewTemplate, placeholder: tuple[str, str] | None = None, waba_id: str | None = None) → TemplateResponse`

'Create Templates' on developers.facebook.com.

- This method requires the WhatsApp Business account ID to be provided when initializing the client.
- To send a template, use `send_template()`.

ATTENTION: In case of an errors, WhatsApp does not return a proper error message, instead, it returns a message of *invalid parameter* with error code of 100. You need to pay attention to the following:

- The template name must be unique.
- The limitiations of the characters in every field (all documented).
- The order of the buttons.

Templates can be created and managed in the WhatsApp Message Templates dashboard.

Example

```
>>> from pywa.types import NewTemplate as NewTemp
>>> wa = WhatsApp(...)
>>> wa.create_template(
...     template=NewTemp(
...         name='buy_new_iphone_x',
...         category=NewTemp.Category.MARKETING,
...         language=NewTemp.Language.ENGLISH_US,
...         header=NewTemp.Text('The New iPhone {15} is here!'),
...         body=NewTemp.Body('Buy now and use the code {WA_IPHONE_15} to get {
...         footer=NewTemp.Footer('Powered by PyWa'),
...         buttons=[
...             NewTemp.UrlButton(title='Buy Now', url='https://example.com/shc
...             NewTemp.PhoneNumberButton(title='Call Us', phone_number='123456
...             NewTemp.QuickReplyButton('Unsubscribe from marketing messages')
...             NewTemp.QuickReplyButton('Unsubscribe from all messages'),
...         ],
...     ),
... )
```

Example for Authentication Template:

```
>>> from pywa.types import NewTemplate as NewTemp
>>> wa = WhatsApp(...)
>>> wa.create_template(
...     template=NewTemp(
...         name='auth_with_otp',
...         category=NewTemp.Category.AUTHENTICATION,
...         language=NewTemp.Language.ENGLISH_US,
...         body=NewTemp.AuthBody(
...             code_expiration_minutes=5,
...             add_security_recommendation=True,
...         ),
...         buttons=NewTemp.OTPButton(
...             otp_type=NewTemp.OTPButton.OtpType.ZERO_TAP,
...             title='Copy Code',
...             autofill_text='Autofill',
...             package_name='com.example.app',
...             signature_hash='1234567890ABCDEF1234567890ABCDEF12345678'
...         )
...     ),
... )
```

Parameters:

- **template** – The template to create.
- **placeholder** – The placeholders start & end (optional, default: `('{', '}')`).
- **waba_id** – The WhatsApp Business account ID (Overrides the client's business account ID).

Returns:

The template created response. containing the template ID, status and category.

WhatsApp.create_flow(*name: str*, *categories: Iterable[FlowCategory | str]*, *clone_flow_id: str | None = None*, *endpoint_uri: str | None = None*, *waba_id: str | None = None*) → str

Create a flow.

- This method requires the WhatsApp Business account ID to be provided when initializing the client.
- New Flows are created in `FlowStatus.DRAFT` status.
- To update the flow json, use `update_flow()`.
- To send a flow, use `send_flow()`.

Parameters:

- **name** – The name of the flow.
- **categories** – The categories of the flow.
- **clone_flow_id** – The flow ID to clone (optional).
- **endpoint_uri** – The URL of the FlowJSON Endpoint. Starting from Flow 3.0 this property should be specified only gere. Do not provide this field if you are cloning a Flow with version below 3.0.
- **waba_id** – The WhatsApp Business account ID (Overrides the client's business account ID).

Example

```
>>> from pywa.types.flows import FlowCategory
>>> wa = WhatsApp(...)
>>> wa.create_flow(
...     name='Feedback',
...     categories=[FlowCategory.SURVEY, FlowCategory.OTHER]
... )
```

Returns:

The flow ID.

Raises:

FlowBlockedByIntegrity – If you can't create a flow because of integrity issues.

WhatsApp.update_flow_metadata(*flow_id: str | int*, ***, *name: str | None = None*, *categories: Iterable[FlowCategory | str] | None = None*, *endpoint_uri: str | None = None*, *application_id: int | None = None*) → bool

Update the metadata of a flow.

Parameters:

- **flow_id** – The flow ID.
- **name** – The name of the flow (optional).
- **categories** – The new categories of the flow (optional).
- **endpoint_uri** – The URL of the FlowJSON Endpoint. Starting from FlowJSON 3.0 this property should be specified only gere. Do not provide this field if you are cloning a FlowJSON with version below 3.0.
- **application_id** – The ID of the Meta application which will be connected to the Flow. All the flows with endpoints need to have an Application connected to them.

Example

```
>>> from pywa.types.flows import FlowCategory
>>> wa = WhatsApp(...)
>>> wa.update_flow_metadata(
...     flow_id='1234567890',
...     name='Feedback',
...     categories=[FlowCategory.SURVEY, FlowCategory.OTHER],
...     endpoint_uri='https://my-api-server/feedback_flow',
...     application_id=1234567890,
... )
```

Returns:

Whether the flow was updated.

Raises:

ValueError – If neither of the arguments is provided.

WhatsApp.update_flow_json(*flow_id: str | int, flow_json: FlowJSON | dict | str | Path | bytes | BinaryIO*) → tuple[bool, tuple[pywa.types.flows.FlowValidationError, ...]]

Update the json of a flow.

Parameters:

- **flow_id** – The flow ID.
- **flow_json** – The new json of the flow. Can be a FlowJSON object, dict, json string, json file path or json bytes.

Examples

```
>>> wa = WhatsApp(...)
```

- Using a Flow object:

```
>>> from pywa.types.flows import *
>>> wa.update_flow_json(
...     flow_id='1234567890',
...     flow_json=FlowJSON(version='2.1', screens=[Screen(...)])
... )
```

- From a json file path:

```
>>> wa.update_flow_json(
...     flow_id='1234567890',
...     flow_json="/home/david/feedback_flow.json"
... )
```

- From a json string:

```
>>> wa.update_flow_json(
...     flow_id='1234567890',
...     flow_json="""{"version": "2.1", "screens": [...]}"""
... )
```

Returns:

A tuple of (success, validation_errors).

Raises:

FlowUpdatingError – If the flow json is invalid or the flow is already published.

WhatsApp.publish_flow(*flow_id: str | int*) → bool

This request updates the status of the Flow to “PUBLISHED”.

- This action is not reversible.
- The Flow and its assets become immutable once published.
- To update the Flow after that, you must create a new Flow. You specify the existing Flow ID as the clone_flow_id parameter while creating to copy the existing flow.

> You can publish your Flow once you have ensured that:

- All validation errors and publishing checks have been resolved.
- The Flow meets the design principles of WhatsApp Flows
- The Flow complies with WhatsApp Terms of Service, the WhatsApp Business Messaging Policy and, if applicable, the WhatsApp Commerce Policy

Parameters:

flow_id – The flow ID.

Returns:

Whether the flow was published.

Raises:

FlowPublishingError – If the flow has validation errors or not all publishing checks have been resolved.

`WhatsApp.delete_flow(flow_id: str | int) → bool`

While a Flow is in DRAFT status, it can be deleted.

Parameters:

flow_id – The flow ID.

Returns:

Whether the flow was deleted.

Raises:

FlowDeletingError – If the flow is already published.

`WhatsApp.deprecate_flow(flow_id: str | int) → bool`

Once a Flow is published, it cannot be modified or deleted, but can be marked as deprecated.

Parameters:

flow_id – The flow ID.

Returns:

Whether the flow was deprecated.

Raises:

FlowDeprecatingError – If the flow is not published or already deprecated.

WhatsApp.get_flow(*flow_id: str | int, invalidate_preview: bool = True, phone_number_id: str | int | None = None*) → FlowDetails

Get the details of a flow.

Parameters:

- **flow_id** – The flow ID.
- **invalidate_preview** – Whether to invalidate the preview (optional, default: True).
- **phone_number_id** – To check that a flow can be used with a specific phone number (optional).

Returns:

The details of the flow.

WhatsApp.get_flows(*invalidate_preview: bool = True, waba_id: str | None = None, phone_number_id: str | int | None = None*) → tuple[pywa.types.flows.FlowDetails, ...]

Get the details of all flows belonging to the WhatsApp Business account.

- This method requires the WhatsApp Business account ID to be provided when initializing the client.

Parameters:

- **invalidate_preview** – Whether to invalidate the preview (optional, default: True).
- **waba_id** – The WhatsApp Business account ID (Overrides the client's business account ID).
- **phone_number_id** – To check that the flows can be used with a specific phone number (optional).

Returns:

The details of all flows.

WhatsApp.get_flow_metrics(*flow_id: str | int, metric_name: FlowMetricName, granularity: FlowMetricGranularity, since: date | str | None = None, until: date | str | None = None*) → dict

Get the metrics of a flow.

Read more at developers.facebook.com.

Parameters:

- **flow_id** – The flow ID.
- **metric_name** – See Available Metrics.
- **granularity** – Time granularity.
- **since** – Start of the time period. If not specified, the oldest allowed date will be used. Oldest allowed date depends on the specified time granularity: DAY - 90 days, HOUR - 30 days.
- **until** – End of the time period. If not specified, the current date will be used.

Returns:

`WhatsApp.get_flow_assets(flow_id: str | int) → tuple[pywa.types.flows.FlowAsset, ...]`

Get all assets attached to a specified Flow.

Parameters:

flow_id – The flow ID.

Returns:

The assets of the flow.

`WhatsApp.register_phone_number(pin: int | str, data_localization_region: str | None = None, phone_id: str | None = None) → bool`

Register a Business Phone Number

- Read more at developers.facebook.com

Example

```
>>> wa = WhatsApp(...)
>>> wa.register_phone_number(password='111111', data_localization_region='US')
```

Parameters:

- **pin** – If your verified business phone number already has two-step verification enabled, set this value to your number's 6-digit two-step verification PIN. If you cannot recall your PIN, you can uptdate it.

- **data_localization_region** – If included, enables local storage on the business phone number. Value must be a 2-letter ISO 3166 country code (e.g. `IN`) indicating the country where you want data-at-rest to be stored.
- **phone_id** – The phone ID to register (optional, if not provided, the client's phone ID will be used).

Returns:

The success of the registration.

`WhatsApp.create_qr_code(prefilled_message: str, image_type: Literal['PNG', 'SVG'] = 'PNG', phone_id: str | None = None) → QRCode`

Create a QR code for a prefilled message.

- Read more at developers.facebook.com

Parameters:

- **prefilled_message** – The prefilled message.
- **image_type** – The type of the image (`PNG` or `SVG`. default: `PNG`).
- **phone_id** – The phone ID to create the QR code for (optional, if not provided, the client's phone ID will be used).

Returns:

The QR code.

`WhatsApp.get_qr_code(code: str, phone_id: str | None = None) → QRCode | None`

Get a QR code.

Parameters:

- **code** – The QR code.
- **phone_id** – The phone ID to get the QR code for (optional, if not provided, the client's phone ID will be used).

Returns:

The QR code if found, otherwise None.

`WhatsApp.get_qr_codes(phone_id: str | None = None) → tuple[pywa.types.others.QRCode, ...]`

Get all QR codes associated with the WhatsApp Business account.

Parameters:

phone_id – The phone ID to get the QR codes for (optional, if not provided, the client's phone ID will be used).

Returns:

Tuple of QR codes.

`WhatsApp.update_qr_code(code: str, prefilled_message: str, phone_id: str | None = None) → QRCode`

Update a QR code.

Parameters:

- **code** – The QR code.
- **prefilled_message** – The prefilled message.
- **phone_id** – The phone ID to update the QR code for (optional, if not provided, the client's phone ID will be used).

Returns:

The updated QR code.

`WhatsApp.delete_qr_code(code: str, phone_id: str | None = None) → bool`

Delete a QR code.

Parameters:

- **code** – The QR code.
- **phone_id** – The phone ID to delete the QR code for (optional, if not provided, the client's phone ID will be used).

Returns:

Whether the QR code was deleted.

`async WhatsApp.webhook_update_handler(update: dict, raw_body: bytes | None = None, hmac_header: str = None) → tuple[str, int]`

Handle the incoming update from the webhook manually.

- Use this function only if you are using a custom server (e.g. Django, aiohttp, etc.).

Example

```python
from aiohttp import web
from pywa import WhatsApp, utils

wa = WhatsApp(..., server=None)

async def my_webhook_handler(req: web.Request) -> web.Response:
    res, status_code = await wa.webhook_update_handler(
        update=await req.json(),
        raw_body=await req.read(),
        hmac_header=req.headers.get(utils.HUB_SIG)
    )
    return web.Response(text=res, status=status_code)

app = web.Application()
app.add_routes([web.post("/my_webhook", my_webhook_handler)])

if __name__ == "__main__":
    web.run_app(app, port=...)
```

Parameters:

- **update** – The incoming update from the webhook (dict).
- **raw_body** – The raw body of the request (to calculate the signature, optional).
- **hmac_header** – The `X-Hub-Signature-256` header (to validate the signature, use `utils.HUB_SIG` for the key).

Returns:

A tuple containing the response and the status code.

async WhatsApp.webhook_challenge_handler(*vt: str*, *ch: str*) → tuple[str, int]

Handle the verification challenge from the webhook manually.

- Use this function only if you are using a custom server (e.g. Django, aiohttp, etc.).

Example

```python
from aiohttp import web
from pywa import WhatsApp, utils

wa = WhatsApp(..., server=None)

async def my_challenge_handler(req: web.Request) -> web.Response:
    challenge, status_code = await wa.webhook_challenge_handler(
        vt=req.query[utils.HUB_VT],
        ch=req.query[utils.HUB_CH],
    )
    return web.Response(text=challenge, status=status_code)

app = web.Application()
app.add_routes([web.get("/my_webhook", my_challenge_handler)])

if __name__ == "__main__":
    web.run_app(app, port=...)
```

Parameters:

- **vt** – The verify token param (utils.HUB_VT).
- **ch** – The challenge param (utils.HUB_CH).

Returns:

A tuple containing the challenge and the status code.

WhatsApp.get_flow_request_handler(*endpoint: str, callback: Callable[[WhatsApp, FlowRequest], FlowResponse | FlowResponseError | dict | None | Awaitable[FlowResponse | FlowResponseError | dict | None]], acknowledge_errors: bool = True, handle_health_check: bool = True, private_key: str | None = None, private_key_password: str | None = None, request_decryptor: Callable[[str, str, str, str, str | None], tuple[dict, bytes, bytes]] | None = None, response_encryptor: Callable[[dict, bytes, bytes], str] | None = None*) → FlowRequestCallbackWrapper

Get a function that handles the incoming flow requests.

- Use this function only if you are using a custom server (e.g. Django, aiohttp, etc.), else use the `WhatsApp.on_flow_request()` decorator.

Example

```python
from aiohttp import web
from pywa import WhatsApp, flows

wa = WhatsApp(..., server=None, business_private_key="...", business_private

async def my_flow_callback(wa: WhatsApp, request: flows.FlowRequest) -> flo
    ...

flow_handler = wa.get_flow_request_handler(
    endpoint="/flow",
    callback=my_flow_callback,
)

async def my_flow_endpoint(req: web.Request) -> web.Response:
    response, status_code = await flow_handler.handle(await req.json())
    return web.Response(text=response, status=status_code)

app = web.Application()
app.add_routes([web.post("/flow", my_flow_endpoint)])

if __name__ == "__main__":
    web.run_app(app, port=...)
```

Parameters:

- **endpoint** – The endpoint to listen to (The endpoint uri you set to the flow. e.g `/feedback_flow`).
- **callback** – The callback function to call when a flow request is received.
- **acknowledge_errors** – Whether to acknowledge errors (The return value of the callback will be ignored, and pywa will acknowledge the error automatically).
- **handle_health_check** – Whether to handle health checks (The callback will not be called for health checks).
- **private_key** – The private key to use to decrypt the requests (Override the global `business_private_key`).
- **private_key_password** – The password to use to decrypt the private key (Override the global `business_private_key_password`).
- **request_decryptor** – The function to use to decrypt the requests (Override the global `flows_request_decryptor`)
- **response_encryptor** – The function to use to encrypt the responses (Override the global `flows_response_encryptor`)

Returns:

A function that handles the incoming flow request and returns (response, status_code).