

WhatsApp Template Management

This documentation covers details of the WhatsApp API documentation

- [REST API components & How to read them](#)
- [WhatsApp Templates - things to know](#)
- [WhatsApp Templates API Management](#)
- [Send Template Messages](#)

REST API components & How to read them

REST APIs are the most prevalent and user-friendly type of APIs you will most likely encounter. As a product manager, you play a pivotal role in working with your engineering team to bring digital solutions to life, and a fundamental part of that collaboration is understanding REST APIs.

In this document, you'll learn how to break down REST APIs into their essential components so you can confidently discuss, dissect, interact with, and debug them.

A brief intro to REST APIs

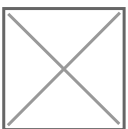
REST APIs are a **set of rules and conventions** that define a standardized way for requests and responses to be structured and exchanged. They are based on the principles of REST, a software architectural style for designing networked applications. You'll also hear the term RESTful APIs in relation to REST APIs.

Here are a couple of examples of RESTful rules and conventions followed by REST APIs (not important to memorize, just to give you an idea of what these rules look like):

1. **Client-server:** Separation of concerns whereby the user interface concerns (client) are separate from the data storage concerns (server).
2. **Stateless:** Each request from the client to the server must contain all of the information necessary to understand and complete the request. The server cannot use previously stored context information on the server.

REST APIs make extensive use of HTTP (Hypertext Transfer Protocol) as the foundation for communication between clients (like web browsers or mobile apps) and servers (where the API is hosted).

The majority of what you need to know about APIs are related to HTTP and its components. We can broadly bucket the components into the API Request and API Response:



Components of an API Request and API Response

API Request

When you make a request to an API, you're essentially saying, "I want to do something or get something from this endpoint." Requests consist of the following components:

1/ API Endpoint

An endpoint is the specific location aka the access point to the API. Think of it as an address for a particular service or resource. These endpoints are usually represented as URLs (Uniform Resource Locators), making them easy to understand. The API endpoint is made up of the "Base URL" plus the "path" of the API.



2/ HTTP Methods

HTTP (Hypertext Transfer Protocol) methods are the actions you can take when interacting with an API. There are 8 request types but 4 are the most commonly used:

- **GET:** This method is like asking for information. When you send a GET request to an API's endpoint, you're **requesting data**, like retrieving weather information for a specific location.
- **POST:** When you send a POST request, you're submitting data to the API which **adds new data to the database**. This could be creating a new user account or adding a comment on a blog post.
- **PUT:** Use the PUT method to **update existing data**. For instance, you might change your profile picture on a social media app.
- **DELETE:** As the name suggests, the DELETE method is for **removing data**. If you want to delete a post or an account, you'd use this method.

You'll likely see all 4 HTTP methods being used at some point, but out of the 4, you'll see the GET and POST method most frequently and those are the two that you should be most familiar with.

Two quick side notes:

- The POST and PUT methods are often used interchangeably. The difference has to do with something called *idempotency* which is very technical and not worth diving into.
- You'll also see the same POST API sometimes used to both *add* new data and *update* existing data. Developers may decide to design the API this way to be more efficient as opposed to creating and managing two separate APIs in the business logic: one POST and the other PUT.

3/ Request Headers

Request headers convey additional information about the API request, such as the format in which the client expects the response (e.g., JSON or XML), authentication credentials, caching instructions, and more.

Here's an example using APIs on Amazon:

```
Accept-Language: en-US
Authorization: Bearer YourAccessTokenHere
Content-Type: application/json
Host: www.amazon.com
Origin: https://www.amazon.com
Referer: https://www.amazon.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117
```

Information in the request header can be used for a number of reasons including:

- **Authenticating the request:** There are various authentication methods, such as API keys, OAuth tokens, and username/password combinations. For example, credentials for your bank account so the API is authorized to make changes to your account.
 - **Caching instructions:** Temporarily storing data in a cache for improved future performance/latency.
 - **Logging:** All systems keep a log of events that occur in a computer system. Information stored in the headers (i.e. location of request origin, browser information, date & time of request, etc..) are logged in log files. The information in the log files can then be used for a variety of reasons: debugging, metrics, and telemetry.
- Accept-Language: en-US
Authorization: Bearer YourAccessTokenHere

4/ Request Body

When you need to send data to the server, you include it in the request body. The data is typically in a structured format like JSON or XML. This is common when creating new records or updating existing ones. The request body is akin to a letter with the information you want to submit.

For example: Amazon has an internal API that adds a new product into the database.

```
POST https://products.amazon.com/api/2017/add
```

```
{
  "product_name": "Smartphone",
  "price": 499.99,
  "description": "A high-quality smartphone with advanced features.",
  "category": "Electronics",
  "availability": true
}
```

API Response

Once you send a API request, you'll receive an API response. This response contains the data or information you requested, as well as metadata about the response itself. It consists of the following components:

1/ Status Code

This is a three-digit number that tells you the outcome of your request. Status codes are like short, standardized messages that quickly convey whether your request was successful, encountered an issue, or faced an error. All status codes are divided into 5 categories but you'll only really encounter 4 of them:

- **2xx: Success** — The API request was successful! *e.g. 200 OK, 201 Created*
- **3xx: Redirection** — The API request has more than one response. *e.g. 301 Moved Permanently, 302 Found*
- **4xx: Client Error** — Some information provided in the request was wrong or missing. *e.g. 400 Bad Request, 401 Unauthorized*
- **5xx: Server Error** — Something went wrong on the server and the request wasn't successful. *e.g. 501 Internal Server Error, 502 Bad Gateway*

Here are some examples of common status codes:

- **200 OK:** Your request was successful, and you'll find the data you wanted in the response.
- **201 Created:** This code typically appears after a successful POST request, indicating that the resource you requested has been created.
- **400 Bad Request:** If the API couldn't understand your request, you might see this code.
- **401 Unauthorized:** When you lack proper authorization or credentials to access the requested data, this code appears.
- **404 Not Found:** This indicates that the endpoint or resource you're looking for doesn't exist.
- **500 Internal Server Error:** If something goes wrong on the other island (the API's server), you might see this code.



Common API status codes

2/ Response Headers

Similar to request headers, response headers provide additional information about the response, such as the type of data you're receiving (e.g., JSON or XML), the server's version, and more.

Here's an example:

```
Access-Control-Allow-Origin: www.amazon.com
Content-Length: 22
Content-Type: application/json
Date: Mon, 30 Oct 2023 16:32:01 GMT
```

3/ Response Body

This is where you'll find the actual data or information you requested. Like the request body, the response body is often in JSON format for easy readability and parsing.

Using the same API example as the one in the request body, here's what a successful response where the product was successfully added to the database might look like. Notice the API response returned a newly created *product_id* uniquely assigned to the added product.

```
200 OK POST https://products.amazon.com/api/2017/add
```

```
{
  "product_id": 12345,
  "message": "Product 'Smartphone' has been successfully added."
```

Conclusion

We've covered the crucial parts of the API and you now have all the information you need to confidently discuss, dissect, interact with, and debug APIs.

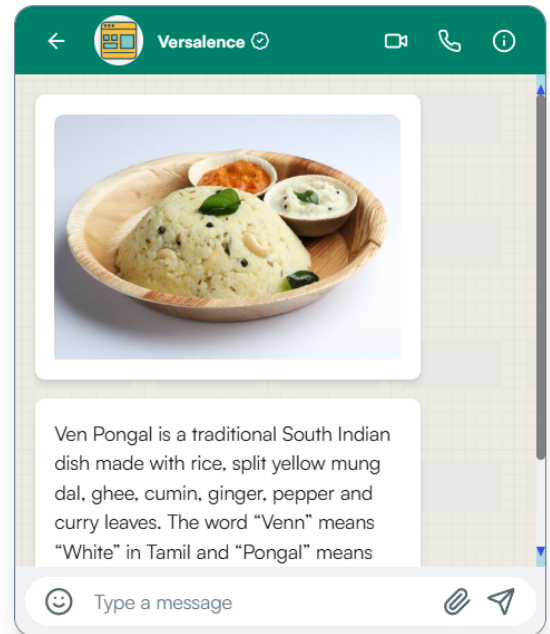


REST API Model

WhatsApp Templates - things to know

Name:	Status:	Category:
<input type="text" value="new_preview6"/>	<input type="text" value="APPROVED"/>	<input type="text" value="Marketing"/>
Type:	Language:	
<input type="text" value="Marketing"/>	<input type="text" value="en"/>	
Header Type:		
<input type="text" value="IMAGE"/>		
Header Image URL:		
<input type="text" value="https://fs.versalence.online/media/e765aafe-45c7-4d08-b8ed-bb3ece3cf78b/temp"/>		
Body Text:		
<input "pongal"="" "to="" "white"="" &="" abundance."="" and="" bubble="" in="" means="" overflow"="" signifies="" tamil="" type="text" value="Ven Pongal is a traditional South Indian dish made with rice, split yellow mung dal, ghee, cumin, ginger, pepper and curry leaves. The word " venn"="" which=""/>		
Footer Text:		
<input type="text" value="N/A"/>		
Quick Reply Buttons:		

Message Preview:



Templates

Templates are used in template messages to open marketing and utility conversations with customers. Unlike free-form messages, template messages are the only type of message that can be sent to customers who have yet to message you, or who have not sent you a message in the last 24 hours.

Templates must be approved before they can be sent in template messages. In addition, templates may be disabled automatically based on customer feedback and engagement. Once disabled, a template cannot be sent in a template message until its quality rating has improved or it no longer violates our [business](#) or [commerce](#) policies.

Creation

Use the template creation API to create templates.

Approval Process

Once you have created your template you can submit it for approval. It can take up to 24 hours for an approval decision to be made. Once a decision has been made, a notification will appear in your Broadcast Templates Manager.

If your message template is approved, its status will be set to ☐ **Approved** and you can begin sending it to customers. If it is rejected, its status will be set to ☐ **Rejected**. The template has to be deleted and wait for 24 hours to reapply again with the same template name.

Multiple templates of the same name can be created by changing the template language

Samples

If your template uses variables you must include sample variable values (media assets, text strings, etc.) with your submission. This makes it easier for us to visualize how your template will appear to customers.

To include a sample with your submission in the WhatsApp Manager, first create your template, add any variables that it requires, and then click the Add Sample button. The preview pane will render any sample media assets or sample text values you provide.

← Back

Create your message template

Choose a category and template to get started.

1 Select a category



Marketing

Promote products and services



Utility

Send utility and update messages



vSecure

Security and verification

If using our APIs to create templates, include the `examples` property for each template component object in your request that uses a variable.

Common Rejection Reasons

Submissions are commonly rejected for the following reasons, so make sure you avoid these mistakes.

- Variable parameters are missing or have mismatched curly braces. The correct format is `{{1}}`.
- Variable parameters contain special characters such as a `#`, `$`, or `%`.
- Variable parameters are not sequential. For example, `{{1}}`, `{{2}}`, `{{4}}`, `{{5}}` are defined but `{{3}}` does not exist.
- Template contains too many variable parameters relative to the message length. You need to decrease the number of variable parameters or increase the message length.
- The message template cannot end with a parameter.
- The message template contains content that violates WhatsApp's Commerce Policy: When you offer goods or services for sale, we consider all messages and media related to your goods or services, including any descriptions, prices, fees, taxes and/or any required legal disclosures, to constitute transactions. Transactions must comply with the [WhatsApp](#)

Commerce Policy.

- The message template contains content that violates the WhatsApp Business Policy: Do not request sensitive identifiers from users. For example, do not ask people to share full length individual payment card numbers, financial account numbers, National Identification numbers, or other sensitive identifiers. This also includes not requesting documents from users that might contain sensitive identifiers. Requesting partial identifiers (ex: last 4 digits of their Social Security number) is OK.
- The content contains potentially abusive or threatening content, such as threatening a customer with legal action or threatening to publicly shame them.
- The message template is a duplicate of an existing template. If a template is submitted with the same wording in the body and footer of an existing template, the duplicate template will be rejected.

WhatsApp Templates API Management

The template APIs allow you to create, delete, and fetch templates

The templates API supports the following methods and are client specific

- GET
- POST
- DELETE

Get all templates

```
curl --location 'https://api.versal.one/<client-id>/templates' \  
--header 'Authorization: Bearer <client-token>'
```

Get a template by name

```
curl --location 'https://api.versal.one/<client-id>/templates?name=order_delivery_1' \  
--header 'Authorization: Bearer <client-token>'
```

Create a template

To create a template - the following components are required

- “
- name - template name [cannot contain Upper case, space, special characters] can contain underscore "_" and a max of 25 characters
 - language - en [English], en_US [English US]. [Click here](#) for the complete list
 - category - Marketing/Utility
 - components - Header, Body, Footer

for more details [check the official documentation](#) on the META website

```
curl --location 'https://api.versal.one/<client-id>/templates' \
--header 'Authorization: Bearer <client-token>' \
--data '{
  "name": "order_delivery_1",
  "language": "en_US",
  "category": "Marketing",
  "components": [
    {
      "type": "HEADER",
      "format": "Text",
      "text": "header text"
    },
    {
      "type": "BODY",
      "text": "This is a {{1}}",
      "example": {
        "body_text": [
          [
            "random text"
          ]
        ]
      }
    },
    {
      "type": "FOOTER",
      "text": "lasadoasi"
    },
    {
      "type": "BUTTONS",
      "buttons": [
        {
          "type": "PHONE_NUMBER",
          "text": "Call",
          "phone_number": "+918897229166"
        }
      ]
    }
  ]
}'
```

```
{
  "message": "Template created successfully",
  "response_text": "{ \"id\": \"1179134613392693\", \"status\": \"APPROVED\", \"category\": \"MARKETING\" }",
  "status": "200"
}
```

Template Components

Templates are made up of four primary components which you define when you create a template: header, body, footer, and buttons. The components you choose for each of your templates should be based on your business needs. The only required component is the body component.

Some components support variables, whose values you can supply when using the Cloud API or On-Premises API to send the template in a template message. If your templates use variables, you must include sample variable values upon template creation.

Headers

Headers are optional components that appear at the top of template messages. Headers support text, media (images, videos, documents), and locations.

All templates are limited to one header component

Text Headers

You can choose to add header text to your message template. The message text in the header component accepts parameters for programmatic configuration.

Text parameters can be configured in one of two formats:

- **Positional** — Pass in an array of parameters that correspond to numeric positions in the body text with examples
 - For example: `"Hello John, your account balance is {{1}}"/> | ["$1,000"]`
- **Named** — Pass in JSON objects that contain a parameter name and examples
 - For example: `{ "param_name": "account_balance", "example": "$1,000" }`

Component Syntax

Add this header component object into the `"components"` object array when calling the `POST <WHATSAPP_BUSINESS_ACCOUNT_ID>/message_templates` endpoint. Substitute the placeholder properties below using the properties table.

```
{
  "type": "HEADER",
  "format": "TEXT",
  "text": "<HEADER_TEXT>",

  "example": {
    "header_text": [
      // You must provide one of the inputs below when the <HEADER_TEXT> string contains parameters
      <POSITIONAL_PARAM_EXAMPLES>
      <BODY_TEXT_NAMED_PARAMS>
    ]
  }
}
```

Properties

Placeholder	Description	Example Value
<code><HEADER_TEXT></code>	<code>String</code> Required Plain text string. Can support 1 parameter. If this string contains parameters, you must include the <code>example</code> property and example parameter values as shown in <code><POSITIONAL_PARAM_EXAMPLES></code> and <code><HEADER_TEXT_NAMED_PARAMS></code> below. 60 character maximum.	<code>"Our Holiday sale starts December 1st!"</code> <code>"Our new sale starts {{1}}"</code> <code>"Our new sale starts {{sale_start_date}}!"</code>
<code><POSITIONAL_PARAM_EXAMPLES></code>	<code>[String]</code> Optional Required when using positional parameters in your header text. Array of <code>String</code> in which each string is meant to illustrate the text likely to be passed in as a parameter during message send time, for example a bank account balance, or a customer name. The number of strings must match the number of variables included in the string.	<code>["December 1st"]</code>
<code><HEADER_TEXT_NAMED_PARAMS></code>	<code>[JSON Object]</code> Optional Required when using named variables in your header text. Array of JSON objects that contain <code>param_name</code> and <code>example</code> strings. <ul style="list-style-type: none"><code>"param_name"</code> — Your custom parameter name. Must be lowercase letters and underscores only.<code>"example"</code> — The illustrative sample text for the parameter.	<code>[{ "param_name": "sale_start_date", "example": "December 1st" }]</code>

Positional Parameter Example

```
{
  "type": "HEADER",
  "format": "TEXT",
  "text": "Our new sale starts {{1}}!",
  "example": {
    "header_text": [
      "December 1st"
    ]
  }
}
```

Named Parameter Example

```
{
  "type": "HEADER",
  "format": "TEXT",
  "text": "Our new sale starts {{sale_start_date}}!",
  "example": {
    "header_text_named_params": [
      {
        "param_name": "sale_start_date",
        "example": "December 1st"
      }
    ]
  }
}
```

Media Headers

Media headers can be an image, video, or a document such as a PDF. All media must be uploaded with the [Resumable Upload API](#). The syntax for defining a media header is the same for all media types.

Syntax

```
{
  "type": "HEADER",
  "format": "<FORMAT>",
  "example": {
    "header_handle": [
      "<HEADER_HANDLE>"
    ]
  }
}
```


Properties

Placeholder	Description	Example Value
<FORMAT>	Indicates media asset type. Set to <code>IMAGE</code> , <code>VIDEO</code> , or <code>DOCUMENT</code> .	<code>IMAGE</code>
<HEADER_HANDLE>	Uploaded media asset handle. Use the Resumable Upload API to generate an asset handle.	<code>4::aW...</code>

Example

```
{
  "type": "HEADER",
  "format": "IMAGE",
  "example": {
    "header_handle": [
      "4::aW..."
    ]
  }
}
```

Body

The body component represents the core text of your message template and is a text-only template component. It is required for all templates.

The message text in the body component accepts parameters for programmatic configuration.

Text parameters can be configured in one of two formats:

- **Positional** — Pass in an array of numbered positional parameters that correspond to numeric positions in the body text with examples
 - For example: `"Hello {{1}}, your account balance is {{2}}" | ["John", "$1,000"]`
- **Named** — Pass in JSON objects that contain a parameter name and examples
 - For example: `{ "param_name": "order_id", "example": "335628" }`

All templates are limited to one body component.

Component Syntax

Add this body component object into the `"components"[]` object array when calling the `POST <WHATSAPP_BUSINESS_ACCOUNT_ID>/message_templates` endpoint. Substitute the placeholder properties below using the properties table.

```
{
  "type": "body",
  "text": "<BODY_TEXT>",
  "example": {
    "body_text": [
      [
        // You must provide one of the inputs below when the <BODY_TEXT> string contains parameters
        <POSITIONAL_PARAM_EXAMPLES>
        <BODY_TEXT_NAMED_PARAMS>
      ]
    ]
  }
}
```

Properties

Placeholder	Description	Example Value
<code><HEADER_TEXT></code>	<code>String</code> Required Plain text string. Can support multiple parameters. If this string contains parameters, you must include the <code>example</code> property and example parameter values as shown in <code><POSITIONAL_PARAM_EXAMPLES></code> and <code><BODY_TEXT_NAMED_PARAMS></code> below. 1024 character maximum.	<code>"Shop our Holiday sale now!"</code> <code>"Shop now through {{1}} and use code {{2}} to get {{3}} off of all merchandise."</code> <code>"Your {{order_id}}, is ready {{customer_name}}"</code>
<code><POSITIONAL_PARAM_EXAMPLES></code>	<code>[String]</code> Optional Required when using positional parameters in your body text. Array of string in which each string is meant to illustrate the text likely to be passed in as a parameter during message send time, for example a bank account balance, or a customer name. The number of strings must match the number of variables included in the string.	<code>["the end of August", "25OFF", "25%"]</code>

Placeholder	Description	Example Value
<BODY_TEXT_NAMED_PARAMS>	<div><i>[JSON Object] Optional</i></div> <p>Required when using named variables in your body text.</p> <p>Array of JSON objects that contain <code>param_name</code> and <code>example</code> strings.</p> <ul style="list-style-type: none"><code>"param_name"</code> — Your custom parameter name. Must be lowercase letters and underscores only.<code>"example"</code> — The illustrative sample text for the parameter.	<pre>[{ "param_name": "order_id", "example": "335628" }, { "param_name": "customer_name", "example": "Shiva" }]</pre>

Positional Parameter Example

```
{
  "type": "BODY",
  "text": "Shop now through {{1}} and use code {{2}} to get {{3}} off of all merchandise.",
  "example": {
    "header_text": [
      "the end of August", "25OFF", "25%"
    ]
  }
}
```

Named Parameter Example

```
{
  "type": "BODY",
  "text": "Your {{order_id}}, is ready {{customer_name}}.",
  "example": {
    "header_text_named_params": [
      {
        "param_name": "order_id",
        "example": "335628"
      },
      {
        "param_name": "customer_name",
        "example": "Shiva"
      }
    ]
  }
}
```

Footer

Footers are optional text-only components that appear immediately after the body component. Templates are limited to one footer component.

Syntax

```
{  
  "type": "FOOTER",  
  "text": "<TEXT>"  
}
```

Properties

Placeholder	Description	Example Value
<code><TEXT></code>	Text to appear in template footer when sent. 60 characters maximum.	<code>Use the buttons below to manage your marketing subscriptions</code>

Example

```
{  
  "type": "FOOTER",  
  "text": "Use the buttons below to manage your marketing subscriptions"  
}
```

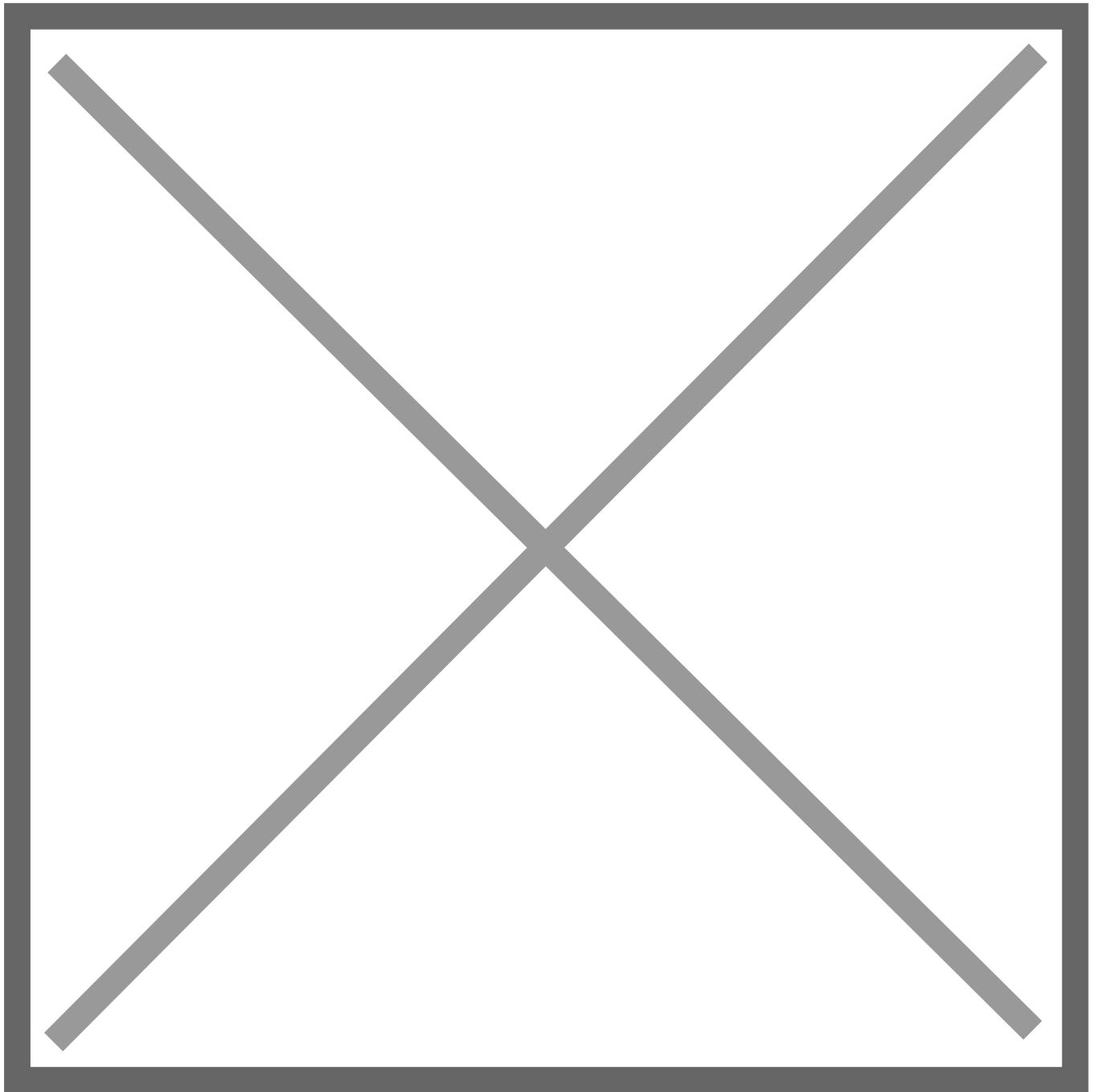
Buttons

Buttons are optional interactive components that perform specific actions when tapped. Templates can have a mixture of up to 10 button components total, although there are limits to individual buttons of the same type as well as combination limits. These limits are described below.

Buttons are defined within a single buttons component object, packed into a single `buttons` array. For example, this template uses a phone number button and a URL button:

```
{
  "type": "BUTTONS",
  "buttons": [
    {
      "type": "PHONE_NUMBER",
      "text": "Call",
      "phone_number": "15550051310"
    },
    {
      "type": "URL",
      "text": "Shop Now",
      "url": "https://www.luckyshrub.com/shop/"
    }
  ]
}
```

If a template has more than three buttons, two buttons will appear in the delivered message and the remaining buttons will be replaced with a **See all options** button. Tapping the **See all options** button reveals the remaining buttons.



Copy Code Buttons

Copy code buttons copy a text string (defined when the template is sent in a template message) to the device's clipboard when tapped by the app user. Templates are limited to one copy code button.

Syntax

```
{  
  "type": "COPY_CODE",  
  "example": "<EXAMPLE>"  
}
```

```
}
```

Properties

Placeholder	Description	Example Value
<EXAMPLE>	String to be copied to device's clipboard when tapped by the app user. Maximum 15 characters.	250FF

Example

```
{  
  "type": "COPY_CODE",  
  "example": "250FF"  
}
```

Delete a template

```
curl --location --request DELETE 'https://api.versal.one/<client-id>/templates?name=harsh45021279' \  
--header 'Authorization: Bearer <client-token>'
```

Send Template Messages

While sending a template message, the template contents are not sent. To send a template message, its media link [public link] & variable values [if any] are sent along with the template name and language

Broadcast with caution - Do not spam. Spamming will lead to the degradation of service and WhatsApp number getting blocked

Sending Message

The template has no parameter

```
curl --location 'https://api.versal.one/<client-id>' \
--header 'Authorization: Bearer <client-token>' \
--header 'Content-Type: application/json' \
--data '{
  "purpose": "sendtemplate",
  "to": <recepient phone number>,
  "template": <template name>,
  "code": <language code>
}'
```

sample data

```
--data '{
  "purpose": "sendtemplate",
  "to": "919999900000",
  "template": "conv_start_hi",
  "code": "en_US"
}'
```

The template has image media

The image typically does not look like a regular parameter while creating the template. However, is a parameter


```

curl --location 'https://api.versal.one/<client-id>' \
--header 'Authorization: Bearer <client-token>' \
--header 'Content-Type: application/json' \
--data '{
  "purpose": "sendtemplate",
  "to": <recepient>,
  "template": "schedule",
  "code": "en",
  "components": [
    {
      "type": "header",
      "parameters": [
        {
          "type": "image",
          "image": {
            "link": "https://cdn.anuhealthyfood.in/images/missyourorders.png"
          }
        }
      ]
    }
  ]
}'

```

The link has to be publically accessible, or else the template won't be delivered to the user

The template has body parameters

```

curl --location 'https://api.versal.one/<client-id>' \
--header 'Authorization: Bearer <client-token>' \
--header 'Content-Type: application/json' \
--data '{
  "purpose": "sendtemplate",
  "to": <recepient>,
  "template": "schedule",
  "code": "en",
  "components": [
    {
      "type": "body",
      "parameters": [

```

```
{
  "type": "TEXT",
  "text": "Hemant Suryavanshi"
},
{
  "type": "TEXT",
  "text": "Blue USB Microphone"
},
{
  "type": "TEXT",
  "text": "12/12/2024"
}
]
}
]
```

In the above example, the template has three body variables. As depicted, only parameters are passed and not the entire body

The template has body parameters and image media

```
curl --location 'https://api.versal.one/<client-id>' \
--header 'Authorization: Bearer <client-token>' \
--header 'Content-Type: application/json' \
--data '{
  "purpose": "sendtemplate",
  "to": <recepient>,
  "template": "schedule",
  "code": "en",
  "components": [
    {
      "type": "header",
      "parameters": [
        {
          "type": "image",
          "image": {
            "link": "https://cdn.anuhealthyfood.in/images/missyourorders.png"
          }
        }
      ]
    }
  ]
}
```

```

    }
  ]
},
{
  "type": "body",
  "parameters": [
    {
      "type": "TEXT",
      "text": "Hemant Suryavanshi"
    },
    {
      "type": "TEXT",
      "text": "Blue USB Microphone"
    },
    {
      "type": "TEXT",
      "text": "12/12/2024"
    }
  ]
}
]
}'

```

In the above example, there is an image and three body parameters in the template

Response

```

{
  "message": "Message sent successfully",
  "response_text":
  "{ \"messaging_product\": \"whatsapp\", \"contacts\": [{ \"input\": \"+16312924377\", \"wa_id\": \"16312924377\" } ], \"messages\": [{ \"id\": \"wamid.HBgLMTYzMTI5MjQzNzcVAgARGBIxNEExODY3MjhGMDNGNDRBNjgA\", \"message_status\": \"accepted\" } ] }",
  "status": "200"
}

```

