**Structured messages**

Facebook allows you to send different types of messages through the bot like plain text, images, text with buttons etc. Also, Facebook now allows a website link to open inside the messenger using the webview. In this guide, we will learn how to use few of these templates on Gupshup's platform.

1. [Button Template](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#ButtonTemplate)
2. [Generic Template](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#GenericTemplate)
3. [Quick Replies](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#QuickReplies)
4. [Share Button](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#ShareButton)
5. [Webview](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#Webview)
6. [Call Button](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#CallButton)
7. [List Template](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#ListTemplate)
8. [Receipt Template](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#ReceiptTemplate)
9. [Message Tags](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages#MessageTags)

Now, let's look at all these, one by one using few examples.

**Button Template**

The Button Template is useful when you want to present simple text with options. It can have multiple options or can have just yes or no.



Facebook imposes certain limitations for Button templates :  
           **Text** : 640 characters and must be UTF-8.  
           **Call-to-action title** : 20 characters.  
           **Call-to-action items** : 3 buttons.

***Example 1 - Creating template with only YES and NO as option***

Let's say you want to ask the user if he likes ice-cream or not. To do so you need to just change the 'type' to 'poll' in the above example and remove the options.   
The JSON would look like -

{

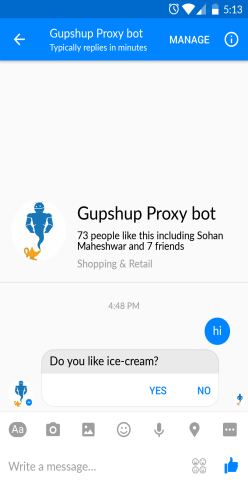
"type": "poll",

"question": "Do you like ice-cream?"

}

This JSON has to be sent as String in the HTTP response and Gupshup will make it work behind the scene.

Here is how this will look on FB messenger



You can also associate a message id to this message , so that you can identify which structured message has been clicked by the user.   
For this, the JSON would look like -

{

"type": "poll",

"question": "Do you like ice-cream?",

"msgid": "poll\_212"

}

Once the user selects one of the options say “Yes” then your bot will receive that option value as the message and you can access this message using ***event.messageobj.text*** and the message id can be accessed using ***event.messageobj.refmsgid***

So if you were using the hosted code on the bot builder then the code to send this structured message and handle the response would be -

if (event.message == "hi"){

var payload = {

"type": "poll",

"question": "Do you like ice-cream?",

"msgid": "poll\_212"

};

context.sendResponse(JSON.stringify(payload));

return;

}

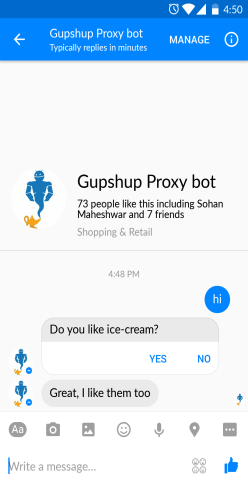
if (event.message == "Yes" && event.messageobj.refmsgid=='poll\_212'){

context.sendResponse("Great, I like them too");

return;

}

The complete flow would look like -



***Example 2 - Creating template with multiple options.***

Let's say you have a restaurant bot,a new user starts chatting with your bot and you want to ask if he/she wants to eat or drink. You also want to include an option for the user to visit your website. To do so you can just send the JSON mentioned below as String through the HTTP response back to the user -

{

"type": "survey",

"question": "What would you like to do?",

"msgid": "3er45",

"options": [

"Eat",

"Drink",

{

"type": "url",

"title": "View website",

"url": "www.gupshup.io"

}

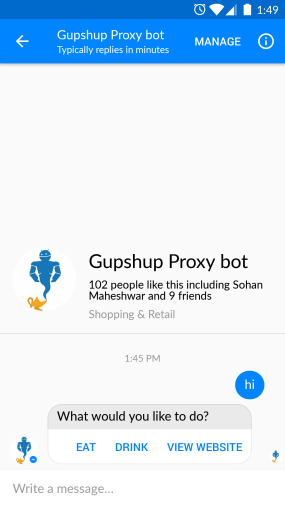
]

}

You can associate a message id to identify this template in the same way as Example 1. 

This JSON has to be sent as String in the HTTP response and Gupshup will make it work behind the scene.

Here is how this will look on FB messenger



If the user clicks on View website then the URL will open into a new browser window or you can configure it to open inside the messenger using ***Webview***.(For Webview scroll down to the last part of this guide).

Once the user selects one of the options say “Eat” then your bot will receive that option value as the message and You can access this message using ***event.messageobj.text*** and the message id can be accessed using ***event.messageobj.refmsgid*** .

So if you were using the hosted code on the bot builder then the code to send this structured message and handle the response would be -

if(event.message=='hi'){

var question = {

"type": "survey",

"question": "What would you like to do?",

"msgid": "3er45",

"options": [

"Eat",

"Drink",

{

"type": "url",

"title": "View website",

"url": "www.gupshup.io"

}

]

};

context.sendResponse(JSON.stringify(question));

return;

}

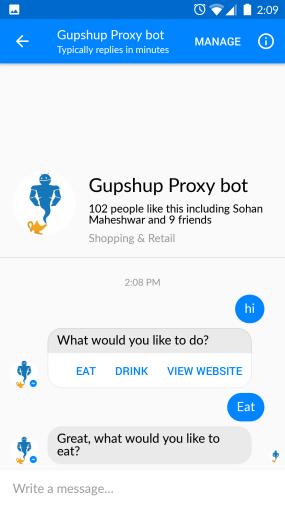
if (event.message == "Eat" && event.messageobj.refmsgid=='3er45'){

context.sendResponse("Great, what would you like to eat?");

return;

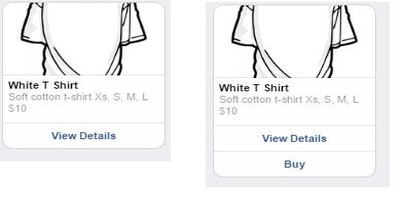
}

The complete flow would look like -



**Generic Template**

The Generic Template can take an image, title, subtitle, description and buttons to request input from the user. This template can support multiple bubbles per message and display them as a horizontally scrollable carousel of items



Facebook imposes certain limitations for Generic template -  
           **Title**: 80 characters.  
           **Subtitle**: 80 characters.  
           **Call-to-action title**: 20 characters.  
           **Call-to-action items**: 3 buttons.  
           **Bubbles per message (horizontal scroll)**: 10 elements.  
           **Image Dimensions**: Image ratio should be 1.91:1.

***Example - Creating a catalogue.***

Let's say you have a menu of online shopping and want to display multiple items in a horizontal list like a carousel to the user. To do so you can use the below JSON in your code -

{

"type": "catalogue",

"imageaspectratio": "horizontal",

"msgid": "cat\_212",

"items": [{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://petersapparel.parseapp.com/img/item100-thumb.png",

"options": [

{

"type": "url",

"title": "View Details",

"url": "http://petersapparel.parseapp.com/img/item100-thumb.png"

},

{

"type": "text",

"title": "Buy"

}

]

},

{

"title": "Grey T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$12",

"imgurl": "http://petersapparel.parseapp.com/img/item101-thumb.png",

"options": [

{

"type": "url",

"title": "View Details",

"url": "http://petersapparel.parseapp.com/img/item101-thumb.png"

},

{

"type": "text",

"title": "Buy"

}]

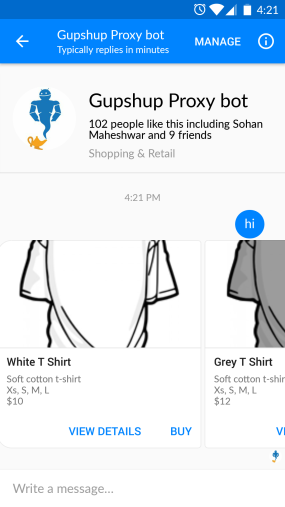
}]

}

You can associate a message id to identify which structured message was clicked by the user in the same way as Button template. 

This JSON has to be sent as String in the HTTP response and Gupshup will make it work behind the scene.

Here is how this will look on FB messenger -



    
In the above JSON you can choose -

1. Not to have an image for one item but have it for another. Just set the parameter to an empty string like {“imgurl”:““} or choose not to send “imgurl” parameter at all.
2. To have different options of buttons for different items. Just make the change in “options” array. You can even choose not to have any buttons by sending empty array of “options” like - { “options”: []}
3. Not to have the subtitle. Just send it as an empty string or do not send the parameter at all.

If the user clicks on View Details then the URL will open into a new browser window or you can configure it to open inside the messenger using ***Webview***.(For Webview scroll down to the last part of this guide).

Once the user selects the option “Buy” for “White T-Shirt” then your bot will receive that option value as the message and you can access this message using ***event.messageobj.text*** and the message id can be accessed using ***event.messageobj.refmsgid*** .

So if you were using the hosted code on the bot builder then the code to send this structured message and handle the response would be -

if(event.message=='hi'){

var catalogue = {

"type": "catalogue",

"imageaspectratio": "horizontal",

"msgid": "cat\_212",

"items": [

{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://petersapparel.parseapp.com/img/item100-thumb.png",

"options":[

{

"type":"url",

"title":"View Details",

"url":"http://petersapparel.parseapp.com/img/item100-thumb.png"

},

{

"type":"text",

"title":"Buy"

}

]

},

{

"title": "Grey T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$12",

"imgurl": "http://petersapparel.parseapp.com/img/item101-thumb.png",

"options":[

{

"type":"url",

"title":"View Details",

"url":"http://petersapparel.parseapp.com/img/item101-thumb.png"

},

{

"type":"text",

"title":"Buy"

}

]

}

]

};

context.sendResponse(JSON.stringify(catalogue));

return;

}

if(event.message=='Buy 1' && event.messageobj.refmsgid=='cat\_212'){

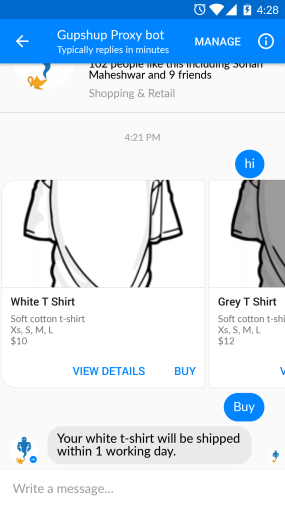
context.sendResponse("Your white t-shirt will be shipped within 1 working day.");

return;

}

In this carousal view the response sent back to the bot consist of the button name and the position of the item in that list. In the above case since we are selecting the first option hence the message returned is "Buy 1".

The complete flow would look like -



Reference for elements inside the generic template JSON -

| **Property Name** | **Description** | **Type** | **Required** |
| --- | --- | --- | --- |
| type | Value must be catalogue | String | Y |
| imageaspectratio | Aspect ratio used to render images specified by imgurl field. Value must be horizontal or square. Defaults to horizontal. | String | N |
| msgid | The identifier for a message | String | N |
| items | List of bubble items | Array | Y |
| title | Bubble title | String | Y |
| subtitle | Bubble subtitle | String | N |
| imgurl | Bubble image | String | N |
| options | Set of buttons that appear as call-to-actions  There are various properties of buttons like -  type - the action type of the button. Can be 'url' or 'text'.  title - Button title. In case of text this title is sent back as resposne. url - For web\_url buttons, this URL is opened in a browser when the button is tapped. | Array | N |

**Default action on Generic template:**

User can trigger the default action by tapping anywhere on the element other than the defined buttons. The default\_action is like a URL Button and contains the same fields except that the title field is not allowed.

This is the sample default action JSON:

"default\_action": {

"type": "web\_url",

"url": "https://peterssendreceiveapp.ngrok.io/view?item=103",

"messenger\_extensions": true,

"webview\_height\_ratio": "tall"

}

**messenger\_extensions** parameter should be only passed if you have enabled messenger\_extension feature for the bot. Find out more in this [guide](https://www.gupshup.io/developer/docs/bot-platform/guide/serverless-webviews-using-gupshup)

The complete sample JSON for the Generic template:(without messenger extension)

{

"type": "catalogue",

"msgid": "cat\_212",

"items": [

{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://petersapparel.parseapp.com/img/item100-thumb.png",

"defaultaction" : {

"type": "url",

"url": "http://logonoid.com/images/batman-logo.png",

"webview\_height\_ratio": "tall"

},

"options": [

{

"type": "url",

"title": "View Details",

"url": "http://logonoid.com/images/batman-logo.png"

},

{

"type": "text",

"title": "Like"

},

{

"type":"element\_share"

}

]

}

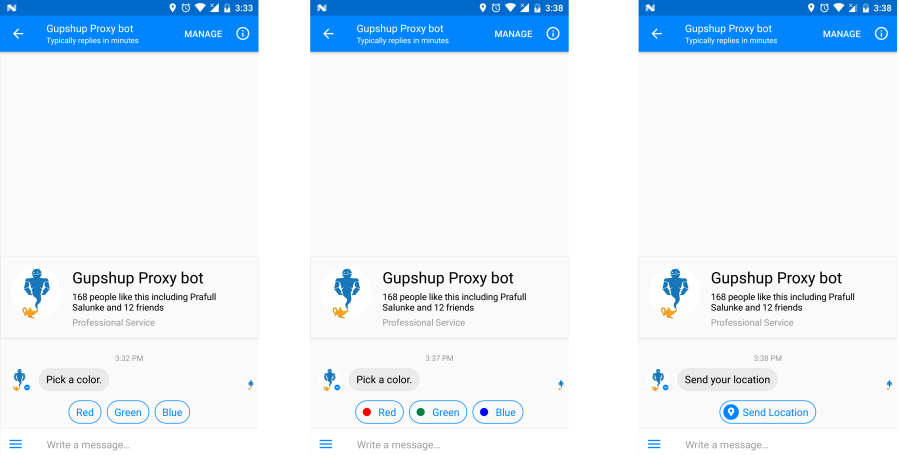
]

}

**Quick Replies**

Quick Replies are buttons with some text with buttons appearing just above the composer. When a button is tapped, the message is sent in the conversation.After a button is tapped, the buttons are dismissed.

Quick Replies contain text. You can also add an image to the quick replies or prompt the user for their location.



**Example -**

Let's say you want to ask a user his/her favorite color or ask the user for his/her location. To do so you can just send one of the below JSON as String through the HTTP response back to the user depending upon your need -

***Sending only text***

{

"type": "quick\_reply",

"content": {

"type": "text",

"text": "What's your favourite color?"

},

"msgid": "qr\_212",

"options": [

"Red",

"Green",

"Yellow",

"Blue"

]

}

***Sending text and image***

{

"type": "quick\_reply",

"content": {

"type": "text",

"text": "What's your favourite color?"

},

"msgid": "qr\_212",

"options": [{

"type": "text",

"title": "Red",

"iconurl": "http://www.iconsplace.com/icons/preview/red/record-256.png"

},

{

"type": "text",

"title": "Green",

"iconurl": "http://www.iconsplace.com/icons/preview/008342/record-256.png"

},

{

"type": "text",

"title": "Blue",

"iconurl": "http://www.iconsplace.com/icons/preview/blue/record-256.png"

}]

}

***Sending location button***

{

"type": "quick\_reply",

"content": {

"type": "text",

"text": "Send your location"

},

"msgid": "qr\_212",

"options": [{

"type": "location"

}]

}

You can associate a message id to identify this quick replies as well. 

This JSON has to be sent as String in the HTTP response and gupshup platform will make it work behind the scene.

In case of text or text with image once the user selects one of the options then your bot will receive that option value as the message and You can access this message using ***event.messageobj.text*** and the message id can be accessed using ***event.messageobj.refmsgid*** .

In case of location sharing button you can view this [guide](https://www.gupshup.io/developer/docs/bot-platform/guide/handling-location-from-user) to understand how to handle the response.

So if you were using the hosted code on the bot builder then the code to send a message with all three quick replies option would be -

if(event.message=='hi'){

var quickreply = {

"type": "quick\_reply",

"content": {

"type": "text",

"text": "Choose one of the option"

},

"msgid": "qr\_212",

"options": [

"Enter address", {

"type": "location"

}, {

"type": "text",

"title": "Skip",

"iconurl": "http://icons.iconarchive.com/icons/visualpharm/must-have/256/Next-icon.png"

}

]

};

context.sendResponse(JSON.stringify(quickreply));

return;

}

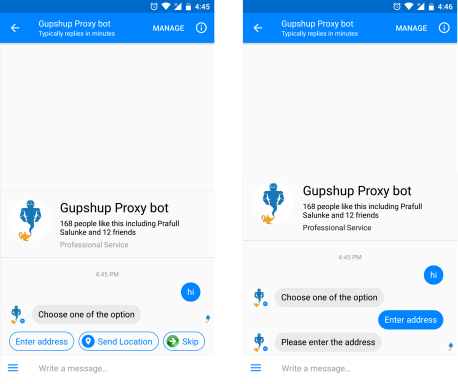
if(event.message=='Enter address'&& event.messageobj.refmsgid=='qr\_212'){

context.sendResponse("Please enter the address");

return;

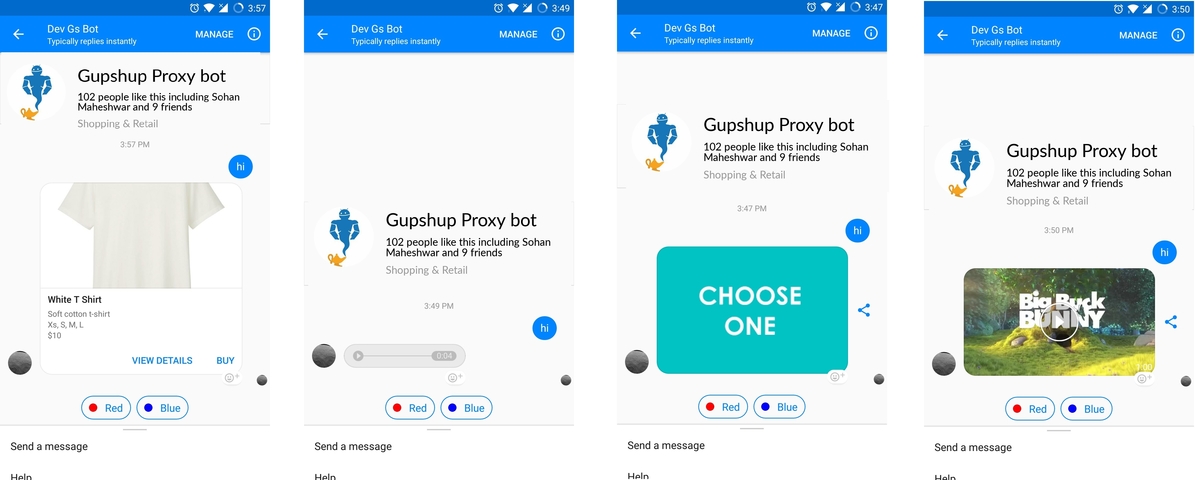
}

The complete flow would look like -



The number of buttons is limited to 11.   
Button title has a 20 character limit.  
Image should be at least 24x24 and will be cropped and resized by FB.

Along with text as the message, quick replies now support attachments too like - image, audio, video or a file.



The sample JSONs for all these type of quick replies are below.   
***Sending image as the message:***

{

"type": "quick\_reply",

"content": {

"type": "image",

"url": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTGn32CwSvlBcY-y3jlxindXL6qAn3UgZiWD0cggIPMtbHTGhN5"

},

"msgid": "rtyertaa",

"options": [{

"type": "text",

"title": "Red",

"iconurl": "http://www.iconsplace.com/icons/preview/red/record-256.png"

},

{

"type": "text",

"title": "Blue",

"iconurl": "http://www.iconsplace.com/icons/preview/blue/record-256.png"

}

]

}

***Sending audio as the message:***

{

"type": "quick\_reply",

"content": {

"type": "audio",

"url": "https://a.clyp.it/v3rprye4.mp3"

},

"msgid": "rtyertaa",

"options": [{

"type": "text",

"title": "Red",

"iconurl": "http://www.iconsplace.com/icons/preview/red/record-256.png"

},

{

"type": "text",

"title": "Blue",

"iconurl": "http://www.iconsplace.com/icons/preview/blue/record-256.png"

}

]

}

***Sending video as the message:***

{

"type": "quick\_reply",

"content": {

"type": "video",

"url": "http://clips.vorwaerts-gmbh.de/big\_buck\_bunny.mp4"

},

"msgid": "rtyertaa",

"options": [{

"type": "text",

"title": "Red",

"iconurl": "http://www.iconsplace.com/icons/preview/red/record-256.png"

},

{

"type": "text",

"title": "Blue",

"iconurl": "http://www.iconsplace.com/icons/preview/blue/record-256.png"

}

]

}

***Sending a file attachment as the message:***

{

"type": "quick\_reply",

"content": {

"type": "file",

"url": "http://unec.edu.az/application/uploads/2014/12/pdf-sample.pdf"

},

"msgid": "rtyertaa",

"options": [{

"type": "text",

"title": "Red",

"iconurl": "http://www.iconsplace.com/icons/preview/red/record-256.png"

},

{

"type": "text",

"title": "Blue",

"iconurl": "http://www.iconsplace.com/icons/preview/blue/record-256.png"

}

]

}

***Sending a Generic template as the message:***

{

"type": "quick\_reply",

"content": {

"type": "catalogue",

"imageaspectratio": "horizontal",

"msgid": "cat\_212",

"items": [{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://media.gq.com/photos/57571f41194831767ce37bec/master/pass/white-tees-reigning-uniqlo.jpg",

"options": [{

"type": "url",

"title": "View Details",

"url": "http://media.gq.com/photos/57571f41194831767ce37bec/master/pass/white-tees-reigning-uniqlo.jpg"

},

{

"type": "text",

"title": "Buy"

}

]

}]

},

"msgid": "rtyertaa",

"options": [{

"type": "text",

"title": "Red",

"iconurl": "http://www.iconsplace.com/icons/preview/red/record-256.png"

},

{

"type": "text",

"title": "Blue",

"iconurl": "http://www.iconsplace.com/icons/preview/blue/record-256.png"

}

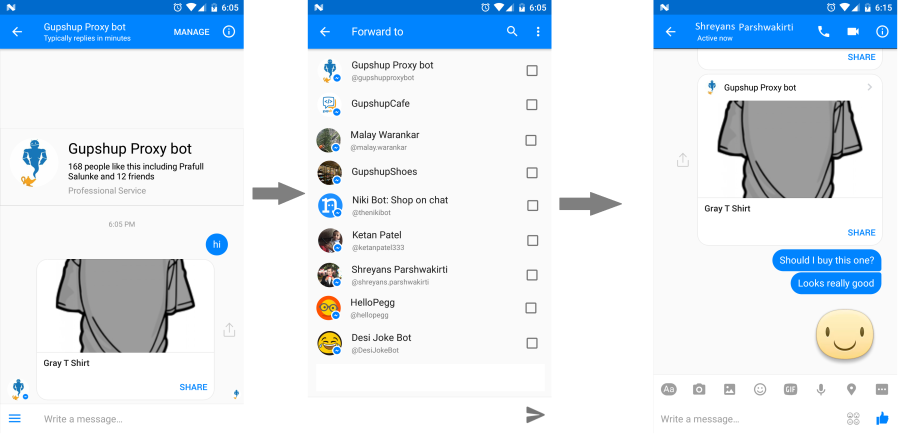
]

}

All the feature of Generic template applies here.

**Share Button**

The Share Button enables users to share message bubbles with their contacts on Facebook.



The user with whom this content is shared can tap on it to start his own conversation with the bot.

You may also wish to specify contents different than the message the “Share” button is attached to. Hence you can change the image, title and sub-title of the shared element and add a URL button. It's like generic template with a maximum of 1 URL button.



To add the sharing feature you can just send the below JSON as String through the HTTP response back to the user -

{

"type": "catalogue",

"imageaspectratio": "horizontal",

"msgid": "cat\_212",

"items": [{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://matchem.com/wp-content/uploads/2015/03/white-shirt.jpg",

"options": [{

"type": "url",

"title": "View Details",

"url": "http://gupshup.io"

}, {

"type": "text",

"title": "Buy"

}, {

"type": "element\_share",

"items": [{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://matchem.com/wp-content/uploads/2015/03/white-shirt.jpg",

"options": [{

"type": "url",

"title": "Chat with me",

"url": "http://m.me/gupshupproxybot"

}]

}]

}]

}]

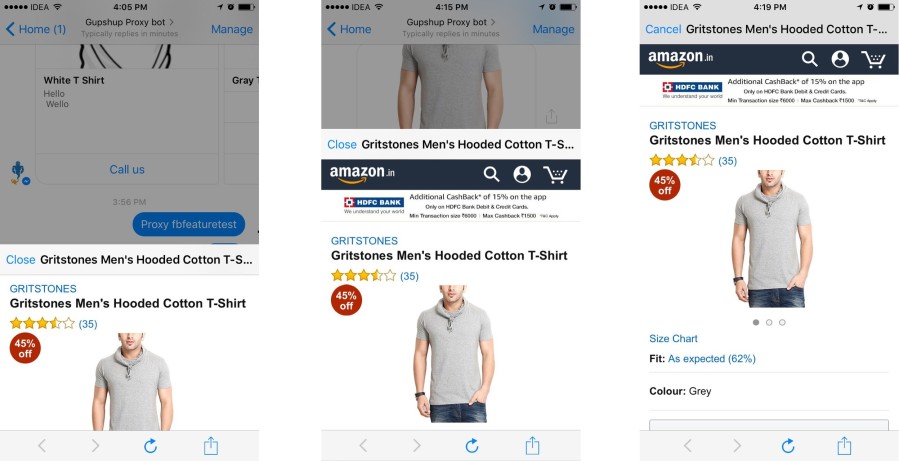
}

The Share Button only works with the Generic Template.  
Only individual message bubbles can be shared.  
The URL botton you add to Shared content can use the webview to show a page or use m.me to deeplink to your bot.

**Webview**

Webview allows you to control the display height of the in-app browser window. This allows you to build more custom experiences that still appear to be part of the Messenger thread.

There are 3 different heights in which webview can be used.



To allow button template or generic template to open the webview , you just need to add one additional parameter called **webview\_height\_ratio** to the button with URL.

***Button Template example***

{

"type": "survey",

"question": "What would you like to do?",

"msgid": "3er45",

"options": [{

"type": "url",

"title": "view website",

"url": "https://gupshup.io",

"webview\_height\_ratio": "compact"

}]

}

***Generic Template example***

{

"type": "catalogue",

"msgid": "6rty",

"items": [{

"title": "Gray T Shirt",

"imgurl": "http://ecx.images-amazon.com/images/I/819Cirp6kPL.\_UL1500\_.jpg",

"options": [{

"type": "url",

"title": "view more",

"url": "http://www.amazon.in/Gritstones-Hooded-T-Shirt-GS60210CWLGMEL-\_Grey\_Large/dp/B00OXR17OE/ref=sr\_1\_6",

"webview\_height\_ratio": "compact"

}]

}]

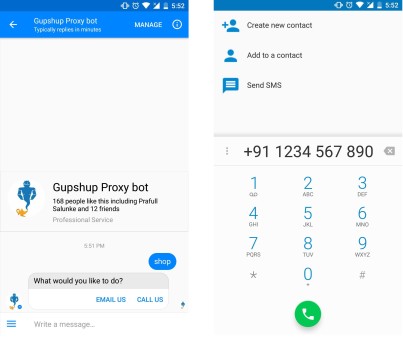
}

The different values that can be assigned for the webview are -

| **Property Name** | **Description** | **Valid values** |
| --- | --- | --- |
| webview\_height\_ratio | Height of the Webview | compact tall full |

**Call Button**

The Call Button can be used to initiate a phone call. Once the user taps on this button he will be taken to the dialer on his/her phone. This button can be used with both Button and Generic Templates.



To add the sharing feature you can just send the below JSON as String through the HTTP response back to the user -

***Button Template example***

{

"type": "survey",

"question": "What would you like to do?",

"msgid": "3er45",

"options": [{

"type": "phone\_number",

"title": "Call us",

"phone\_number": "+911234567890"

}]

}

***Generic Template example***

{

"type": "catalogue",

"msgid": "6rty",

"items": [{

"title": "Gray T Shirt",

"imgurl": "http://ecx.images-amazon.com/images/I/819Cirp6kPL.\_UL1500\_.jpg",

"options": [{

"type": "phone\_number",

"title": "Call for more information",

"phone\_number": "+911234567890"

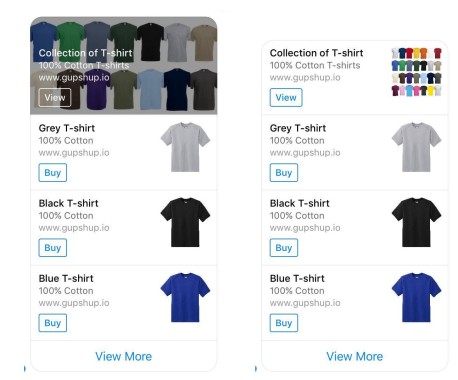
}]

}

**List Template**

The List template allows you to present a set of items vertically. It can be rendered in two different ways.

One way is to make the first item of the list more prominent like a cover image and the other way is to keep all the items identical.



There are few limitations imposed by Facebook for this template:

1. You need to send at least 2 elements with a maximum of 4 elements.
2. Adding a button to each element is optional.
3. You may only have up to 1 button per element.You may have up to 1 global button.
4. ***title*** has 80 character limit. Note that it can get truncated on the client if the element takes too many lines.
5. ***subtitle*** has 80 character limit. Note that it can get truncated on the client if the element takes too many lines.

**Sample JSON for List template with items identical in the list**

{

"type": "list",

"topElementStyle": "compact",

"msgid": "list123",

"items": [

{

"title": "Collection of T-shirt",

"subtitle": "100% Cotton T-shirts",

"imgurl": "https://pbs.twimg.com/media/CO2l-3oUYAAMV5S.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "View",

"type": "url",

"url": "https://www.gupshup.io",

"webview\_height\_ratio": "tall"

}

]

},

{

"title": "Grey T-shirt",

"subtitle": "100% Cotton",

"imgurl": "http://site.topdogshirts.com/images/blank/t473silver.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "Buy",

"type": "text"

}

]

},

{

"title": "Black T-shirt",

"subtitle": "100% Cotton",

"imgurl": "http://site.topdogshirts.com/images/blank/t473black.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "Buy",

"type": "text"

}

]

},

{

"title": "Blue T-shirt",

"subtitle": "100% Cotton",

"imgurl": "http://site.topdogshirts.com/images/blank/t473trueroyal.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "Buy",

"type": "text"

}

]

}

],

"globalButtons": [

{

"type": "text",

"title": "View More"

}

]

}

To get the List template with a cover image, just remove the element ***topElementStyle*** from the above JSON.

This JSON has to be sent as String in the HTTP response and Gupshup will make it work behind the scene.

The buttons for the items in the list can be of type ***text*** or ***URL*** the way it was in ***Generic template*** above.

If the user clicks on the button of an item which is of type URL then the URL assigned to that button will open inside the messenger's webview.(To know more about webview scroll up to the webview section).

If the user clicks on the button with type as 'text' then your bot will receive the value of that button as the message and you can access this message using ***event.messageobj.text*** and the message id can be accessed using ***event.messageobj.refmsgid***.

Let's say a user clicks on the buy button of the second item in the list. So, if you are using the IDE Bot Builder then the code to handle the response would be:

if(event.message=='hi'){

var list ={

"type": "list",

"topElementStyle": "compact",

"msgid": "list123",

"items": [

{

"title": "Collection of T-shirt",

"subtitle": "100% Cotton T-shirts",

"imgurl": "https://pbs.twimg.com/media/CO2l-3oUYAAMV5S.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "View",

"type": "url",

"url": "https://www.gupshup.io",

"webview\_height\_ratio": "tall"

}

]

},

{

"title": "Grey T-shirt",

"subtitle": "100% Cotton",

"imgurl": "http://site.topdogshirts.com/images/blank/t473silver.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "Buy",

"type": "text"

}

]

},

{

"title": "Black T-shirt",

"subtitle": "100% Cotton",

"imgurl": "http://site.topdogshirts.com/images/blank/t473black.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "Buy",

"type": "text"

}

]

},

{

"title": "Blue T-shirt",

"subtitle": "100% Cotton",

"imgurl": "http://site.topdogshirts.com/images/blank/t473trueroyal.jpg",

"defaultAction": {

"type": "url",

"title": "view details",

"webview\_height\_ratio": "tall",

"url": "https://www.gupshup.io"

},

"options": [

{

"title": "Buy",

"type": "text"

}

]

}

],

"globalButtons": [

{

"type": "text",

"title": "View More"

}

]

};

context.sendResponse(JSON.stringify(list));

return;

}

if(event.messageobj.text=='Buy 2' && event.messageobj.refmsgid=='list123'){

context.sendResponse("Your Grey t-shirt will be shipped to you.");

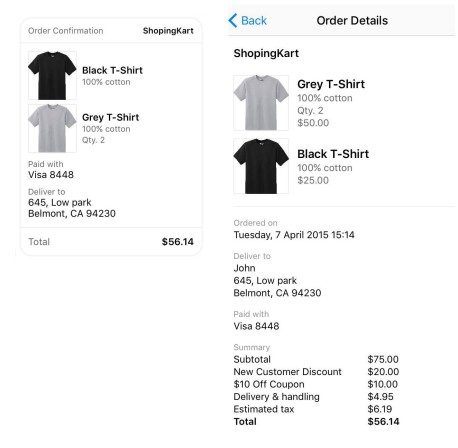
return;

}

In this List template, the response sent back to the bot consists of the button name and the position of the item in that list. In the above case the message returned is "Buy 2", because the item is in the second position of the list.

**Receipt Template**

Receipt Template is used to send an order confirmation, with the transaction summary and description for each item.



    
**Sample JSON for this template**

{

"type": "receipt",

"recipient\_name": "John",

"merchant\_name": "ShopingKart",

"order\_number": "342121213",

"currency": "USD",

"payment\_method": "Visa 8448",

"timestamp": "1428444852",

"order\_url": "http://petersapparel.parseapp.com/order?order\_id=123456",

"elements": [

{

"title": "Grey T-Shirt",

"subtitle": "100% cotton",

"quantity": 2,

"price": 50,

"currency": "USD",

"image\_url": "http://site.topdogshirts.com/images/blank/t473silver.jpg"

},

{

"title": "Black T-Shirt",

"subtitle": "100% cotton",

"quantity": 1,

"price": 25,

"currency": "USD",

"image\_url": "http://site.topdogshirts.com/images/blank/t473black.jpg"

}

],

"address": {

"street\_1": "645, Low park",

"street\_2": "",

"city": "Belmont",

"postal\_code": "94230",

"state": "CA",

"country": "US"

},

"summary": {

"subtotal": 75,

"shipping\_cost": 4.95,

"total\_tax": 6.19,

"total\_cost": 56.14

},

"adjustments": [

{

"name": "New Customer Discount",

"amount": 20

},

{

"name": "$10 Off Coupon",

"amount": 10

}

]

}

This JSON has to be sent as String in the HTTP response and Gupshup will make it work behind the scene.

To know about the mandatory and non-mandatory parameters, please visit Facebook's documentation on [Receipt template](https://developers.facebook.com/docs/messenger-platform/send-api-reference/receipt-template).

[Here is the sample code](https://github.com/GupshupBot/sampleCode/blob/master/Sample%20Code%20-%20Structured%20Messages%20on%20FB.js)

**Message Tags**

Message tags are introduced by Facebook allowing your bot to send messages outside the 24+1 window, for a limited number of use cases, per [Messenger Platform policy](https://developers.facebook.com/docs/messenger-platform/policy-overview).

To know more about the list of these tags please read this [guide](https://developers.facebook.com/docs/messenger-platform/send-api-reference/tags) by Facebook.

Only generic template messages can be sent with tags other than ISSUE\_RESOLUTION. ISSUE\_RESOLUTION tag can be used with either generic template messages or text messages.

***Sample JSON for a message with tag:***

{

"type": "catalogue",

"msgid": "cat\_212",

"tag" : "SHIPPING\_UPDATE",

"items": [

{

"title": "White T Shirt",

"subtitle": "Soft cotton t-shirt \nXs, S, M, L \n$10",

"imgurl": "http://petersapparel.parseapp.com/img/item100-thumb.png",

"defaultaction" : {

"type": "url",

"title": "View Details",

"url": "http://logonoid.com/images/batman-logo.png"

},

"options": [

{

"type": "url",

"title": "View Details",

"url": "http://logonoid.com/images/batman-logo.png"

},

{

"type": "text",

"title": "Like"

},

{

"type":"element\_share"

}

]

}

]

}

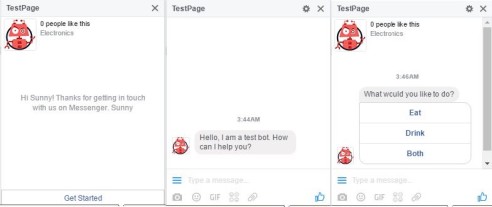
**Get Started Button and Persistent Menu**

Facebook has different settings on Messenger to improve the user-experience of your chatbot. These settings apply to each page individually. In this document, we will learn about two such settings:

1. [Get Started Button](https://www.gupshup.io/developer/docs/bot-platform/guide/get-started-button-persistent-menu-fb#GetStartedButton)
2. [Persistent Menu](https://www.gupshup.io/developer/docs/bot-platform/guide/get-started-button-persistent-menu-fb#PersistentMenu)

**Get Started Button**

The Get Started button is only rendered the first time the user interacts with the page on Messenger. The welcome screen on Messenger displays a Get Started button. When this button is tapped, you can present a personalized message to greet the user or present buttons to prompt him or her to take an action.



***How to set the response of Get Started Button***?

You will be provided with an event called ***startchattingevent*** when someone clicks on the Get Started Button. This can be fetched from the ***messageobj*** parameter.

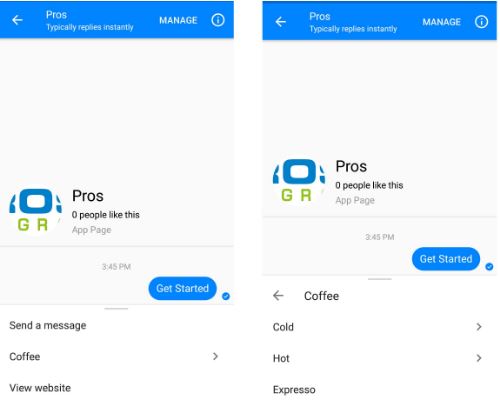
1. If you're using the hosted code then in the ***EventHandler*** use the code snippet below to setup the Get Started message.
2. function EventHandler(context, event) {
3. if(event.messageobj.text=='startchattingevent'&& event.messageobj.type=='event'){
4. var button = {
5. "type": "survey",
6. "question": "What would you like to do?",
7. "options": ["Eat", "Drink", "Both"],
8. "msgid": "gt\_212"
9. }
10. context.sendResponse(JSON.stringify(button));
11. }

This code snippets implements the button template. In the same way you can send either plain text or other structured messages as defined in this [guide](https://www.gupshup.io/developer/docs/bot-platform/guide/structured-messages)

1. If you are using the [callback](https://www.gupshup.io/developer/docs/bot-platform/guide/building-bot-outside-bot-builder-tool) then you can use the parameter ***messageobj*** and parse the JSON to get the event name.

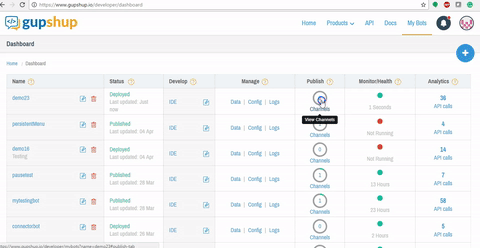
**Persistent Menu**

The Persistent Menu is always available to the user and brings them directly into the flows in your bot. Persistent Menu can also be used to stop users from sending any text messages to the bot.



***How to setup the Persistent Menu***?

You can use the Publish tab in the IDE Bot Builder to set the Persistent Menu.



***Sample JSON***

{

"disableinput" : false,

"menu": [

{

"title": "Coffee",

"type": "nested",

"menu": [

{

"title": "Cold",

"type": "nested",

"menu": [

{

"title": "Chocolate Mocha",

"type": "text"

},{

"title": "Java Chip",

"type": "text"

}

]

},

{

"title": "Hot",

"type": "nested",

"menu": [

{

"title": "Cappuccino",

"type": "text"

},

{

"title": "Caffe Latte",

"type": "text"

}

]

},

{

"title": "Expresso",

"type": "text",

}

]

},

{

"title": "View website",

"type": "url",

"url": "https://www.gupshup.io/developer/home",

"webview\_height\_ratio": "full"

}

]

}

**Things to remember:**   
1. You can have a maximum of 3 hierarchical levels of menus.  
2. A Persistent Menu is limited to 3 items for the top level, and 5 items for any submenus.  
3. To create a multi level menu change the type to "nested" for the desired menu item and add a new parameter "menu" which will be an array of items.  
4. Title of a menu item is limited to 30 characters.  
5. **disableinput** has to be set as "true" in the above JSON to disable the text input in the messenger.  
6. You can have web\_url buttons in the menu as well. Ex: The "View website" item in the above example.  
7. Bot menus are cached locally, but updates are fetched periodically by Facebook. If you update the menu while testing, you can force this fetch to happen by deleting the bot's thread and then beginning a new one.

***Handling user response***:

When someone clicks on one of the items on the menu which is of type ***text*** then the title of that item is sent back as the message.

In the above JSON sample if a user clicks on Expresso then here is what the ***messageobj*** will contain -

messageobj={"referralParam":"","refmsgid":"persistent\_menu","text":"Expresso","type":"msg"}

The ***refmsgid*** will always be returned as “persistent\_menu” if a user selects an option from the Persistent Menu.

In the IDE Bot builder this is how you can handle the response from the user:

if(event.messageobj.refmsgid=='persistent\_menu'&& event.message.toLowerCase()=='expresso'){

context.sendResponse("Sure, I will add Expresso to your order");

return;

}

**Referral and Parametric Codes**

There can be different entry points to your bot with capabilities to trigger different functionalities. Let's look at 2 such options:

1. [Referral](https://www.gupshup.io/developer/docs/bot-platform/guide/referral-on-facebook#ReferralParam)
2. [Parametric Codes](https://www.gupshup.io/developer/docs/bot-platform/guide/referral-on-facebook#ParametricCodes)

**Referral**

One of the ways to interact with the bot on Facebook messenger is to follow the m.me link of the bot. This m.me link is the shortened URL that redirects a user directly to the chat window in messenger. Example - **https://m.me/thebot**

You can now add an additional parameter to this m.me link which is then sent to the bot. Example - **https://m.me/thebot?ref=myrefparam**

Whenever the m.me link is used with the referral paramerter for the bot on Gupshup platform, the bot receives the value of that parameter in the ***messageobj*** 's ***referralParam*** element.

**Sample JSON of the parameters received by the bot**

{

"botname": "thebot",

"contextobj": {

"botname": "thebot",

"channeltype": "fb",

"contextid": "12961285222",

"contexttype": "p2p"

},

"messageobj": {

"referralParam": "myrefparam",

"text": "startchattingevent",

"type": "event"

},

"senderobj": {

"channelid": "12961285222",

"channeltype": "fb",

"display": "John Doe",

"subdisplay": ""

},

"channel": "fb",

"sender": "12961285222",

"message": "startchattingevent"

}

If you are using the Gupshup's IDE Bot Builder to develop the bot then you will receive this JSON in ***EventHandler*** function.

The sample code to handle this is:

function EventHandler(context, event) {

if(event.messageobj.referralParam=='myrefparam'&& event.messageobj.type=='event'){

context.sendResponse("Hello, I am a bot. How can I help");

}

}

**Important notes :**   
1. To receive this referral parameter **messaging\_referral** has to be checked in the webhook settings for your app on developers.facebook.com.  
2. This ref parameter is treated as a string. If you want to send more complex, structured data, it must be encoded. The maximum length is 2,083 characters.  
3. If a page hasn't set up a "Get Started" button and the user has no prior conversation with the bot, the referral will not be conveyed to the bot. Therefore, to use this feature you must set a Get started for your page.

**Parametric Codes**

Parametric Codes can be scanned to instantly link the user to your bot. You can stick these codes on fliers, ads, or anywhere else where you want your bot to be used by people. This is different than the static code because you can pass in a ref parameter which is sent to the bot as a postback when the code is scanned.

You can also generate multiple parametric codes with different ref parameters for a single bot depending upon the use case.



***How to generate a parametric code?***

You will have to call the graph API of facebook to do so.

curl -X POST -H "Content-Type: application/json" -d '{

"type": "standard",

"data": {

"ref":"billboard-ad"

},

"image\_size": 1000

}' "https://graph.facebook.com/v2.6/me/messenger\_codes?access\_token=<ACCESS\_TOKEN>"

Read this [documentation](https://developers.facebook.com/docs/messenger-platform/messenger-code) for more information.

***How to handle the response on Gupshup?***

When the parametric code is scanned, the value of the “ref” parameter is sent to the ***EventHandler*** of the IDE Bot Builder code as a **referral** parameter. Hence you can define your logic based on the value.

***Sample JSON received by the bot:***

{

"botname": "xyzbankbot",

"contextobj": {

"botname": "xyzbankbot",

"channeltype": "fb",

"contextid": "16837306669484",

"contexttype": "p2p"

},

"messageobj": {

"referralParam": "billboard-ad",

"text": "startchattingevent",

"type": "event"

},

"senderobj": {

"channelid": "16837306669484",

"channeltype": "fb",

"display": "John",

"subdisplay": ""

},

"channel": "fb",

"sender": "16837306669484",

"message": "startchattingevent"

}

Your bot on Gupshup platform should be published on Facebook

***Example:***

If you created a code with

"data": {

"ref":"xyzbank"

}

Then you can handle this in the code as:

function EventHandler(context, event) {

if(event.messageobj.referralParam=='xyzbank'&& event.messageobj.type=='event'){

context.sendResponse("Hello "+event.senderobj.display+". I am the XYZbank bot. I can help you with many things but first please enter your phone number");

}

}

# Messenger Opt-in Plugin

The opt-in plugin is basically a callback that occurs when the [Send to messenger](https://developers.facebook.com/docs/messenger-platform/plugin-reference/send-to-messenger) button has been tapped.

This plugin can be used to validate the user and to start the conversation with the user.

While setting up the send to messenger plugin a field called “**data-ref**” is passed which is then returned in the opt-in callback as an event which can be then used to authenticate the user and start the chat.

Gupshup has built support for this opt-in feature on it's platform.

This will only work for the bots published on FB channel. If the bot is not yet FB approved then only admins/developers/testers of the Facebook app can access this feature.

If the **send to messenger** button is clicked then with Opt-in feature avaialbe on the platform these parameters are sent to the bot:

{

"botname": "optinBot",

"contextobj": {

"botname": "optinBot",

"channeltype": "fb",

"contextid": "1246056548800670",

"contexttype": "p2p"

},

"messageobj": {

"referralParam": "WEBSITE\_USER",

"text": "botoptinevent",

"type": "event"

},

"senderobj": {

"channelid": "1246056548800670",

"channeltype": "fb",

"display": "John doe",

"subdisplay": ""

},

"channel": "fb",

"sender": "1246056548800670",

"message": "botoptinevent"

}

If you are using the “**Link your bot**” functionality then these fields will be sent to the bot on the callback URL set on Gupshup platform.

If you are using the hosted IDE of Gupshup platform then you get **EventHandler** function from the bot code is called. You can put a check for the **botoptinevent** and write the logic as required.

In the JSON above the value of **referralParam** is fetched from the Opt-in callback which gets it from “**data-ref**” field of the **send to messenger** button.

Let's say you want to send a welcome message when someone clicks on the **send to messenger** button. The code to handle this in the hosted IDE would be -

function EventHandler(context, event) {

if(event.messageobj.text=="botoptinevent"&& event.messageobj.referralParam=="WEBSITE\_USER"){

context.sendResponse("Welcome! Thanks for visiting our website. You can ask any question related to website to me.");

return;

}

}

Hence steps you need to follow in order to utilize this feature on Gupshup are:   
   1.   **Publish your bot on Facebook**: You need to go to “My bots” section and then select the channels option under publish tab for the corresponding bot and then follow the steps.

   2.   **Handle the code**: In your bot code you need to handle the event trigger as explained above.

   3.   **Create the send to messenger button**: Create the send to messenger button on your website or any other HTML as required. Use the below script and plugin code to do so.

**Script**: T go into head or body of the HTML -

<script>

window.fbAsyncInit = function() {

FB.init({

appId: "APP\_ID",

xfbml: true,

version: "v2.6"

});

};

(function(d, s, id){

var js, fjs = d.getElementsByTagName(s)[0];

if (d.getElementById(id)) { return; }

js = d.createElement(s); js.id = id;

js.src = "//connect.facebook.net/en\_US/sdk.js";

fjs.parentNode.insertBefore(js, fjs);

}(document, 'script', 'facebook-jssdk'));

</script>

**Plugin code**: To go into the body of the HTML

<div class="fb-send-to-messenger"

messenger\_app\_id="APP\_ID"

page\_id="PAGE\_ID"

data-ref="PASS\_THROUGH\_PARAM"

color="blue"

size="standard">

</div>

**messenger\_app\_id**: The ID of the app created on develope.messenger.com while publishing the bot.

**page\_ide**: The id of the page where you published your bot.

**NOTE** : Value of **data-ref** should be one word, spacing is not allowed.

There are more limitations which you can view from [facebooks documentation](https://developers.facebook.com/docs/messenger-platform/plugin-reference/send-to-messenger)

# Build, deploy & test bots on Facebook Messenger

Our [earlier guides](https://www.gupshup.io/developer/docs/bot-platform/guide/build-a-simple-bot-in-2-minutes) demonstrated the creation of a bot using Gupshup's Bot Builder tool. Using the methods mentioned in the above guides, let us create a simple bot that asks the user for his/her favourite publication and then prints the day's top stories from that publication. We will then publish this bot to Facebook Messenger.

Please do read the previous guides, before proceeding.

#### 1.   Build:

Create a new bot using the Bot Bulider. In the messageHandler() method, ask for the user's favourite publication.

if(event.message.toLowerCase() == "hi") {

context.sendResponse("Hey there " + event.sender + " Do you prefer reading Wired or Techcrunch?");

}

Lets us set the user's preferences based on the answer given.

else if((event.message.toLowercase() == "wired") || (event.message.toLowercase() == "techcrunch")) {

setPreference(event.message);

}

We're going to use [persistence](https://www.gupshup.io/developer/docs/bot-platform/guide/adding-advanced-persistence) to store the user's preference.

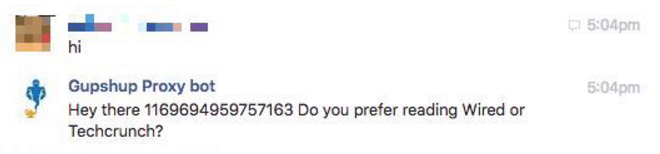
function setPreference (pref) {

context.simpledb.roomleveldata.publication = pref;

}

#### 2.   Test:

Let's use the [Gupshup Proxy Bot](https://www.gupshup.io/developer/demobots) to test out the bot.



Once the user's preference is set, make an http call to the publication's RSS feed. Here, we're using Google's RSS feed to get the top story of the day.   
Here's how it will look:

function setPreference (pref) {

context.simpledb.roomleveldata.publication = pref;

context.simplehttp.makeGet('https://ajax.googleapis.com/ajax/services/feed/load?v=2.0&q=http://www.' + event.message + '.com/feed/&num=1');

}

In our HttpResponseHandler() handler, parse the response from the RSS feed and display the news article. This is how the method will look:

function HttpResponseHandler(context, event) {

var respJson = JSON.parse(event.getresp);

var stories = respJson.responseData.feed.entries;

var resp = "";

for (var i = 0; i<1; i++) {

resp = resp + stories[i].title + "\n" + stories[i].link + "\n";

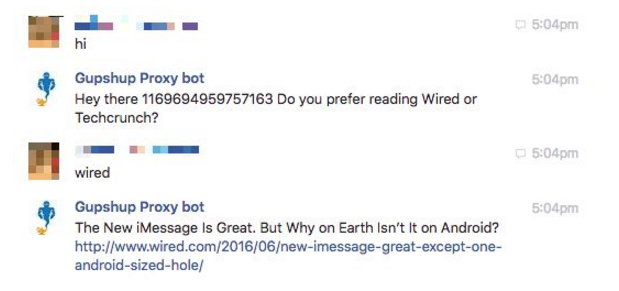
}

resp = resp.replace("&nbsp", "");

context.sendResponse(resp);

}

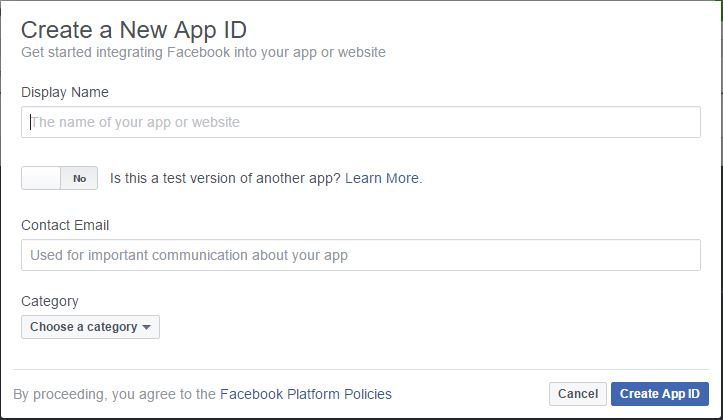
Here is an image of the bot in action



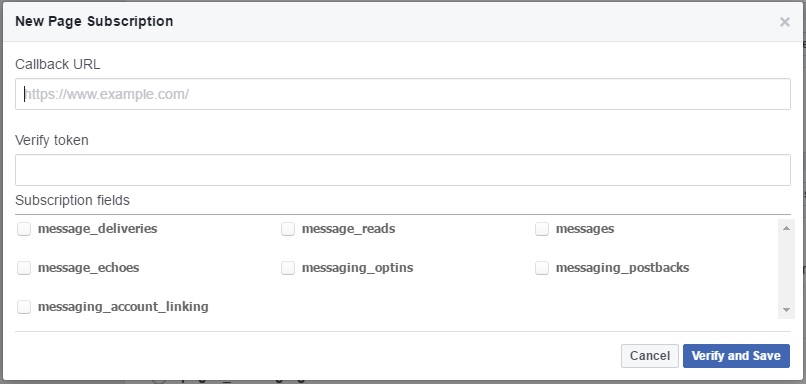
Your bot is now ready to be [published to any messaging channel.](https://www.gupshup.io/developer/docs/bot-platform/guide/publish-bot-to-channels)

#### 3.   Deploy:

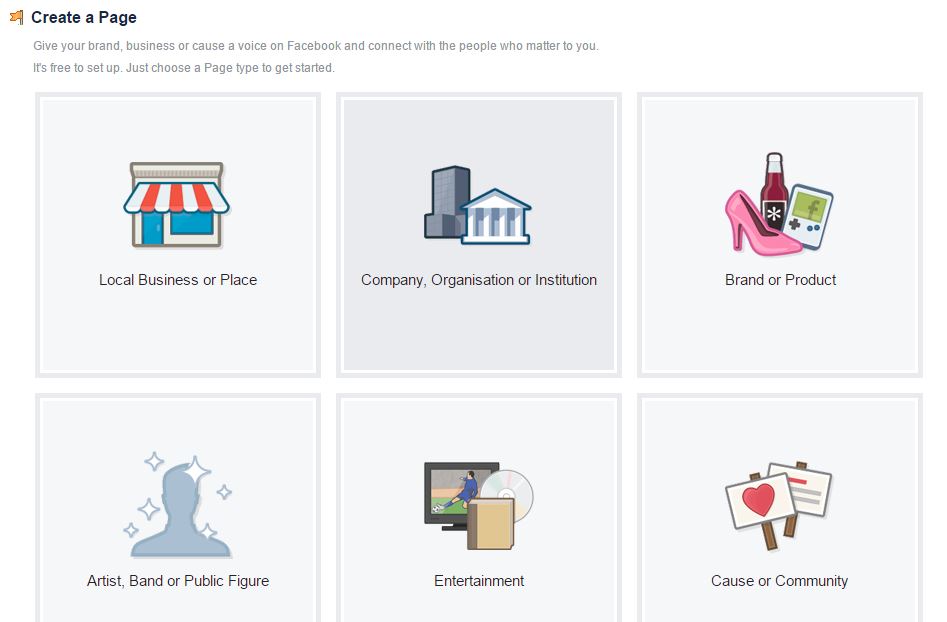
Now publish your bot to Facebook Messenger using the Bot Builder's publish functionality.   
Go to [Facebook developer portal](https://www.gupshup.io/developer/docs/bot-platform/guide/developers.facebook.com) and login. Click on the My Apps tab at the top-right of your screen and 'Add a New App'.   
The Facebook App can remain in sandbox mode.



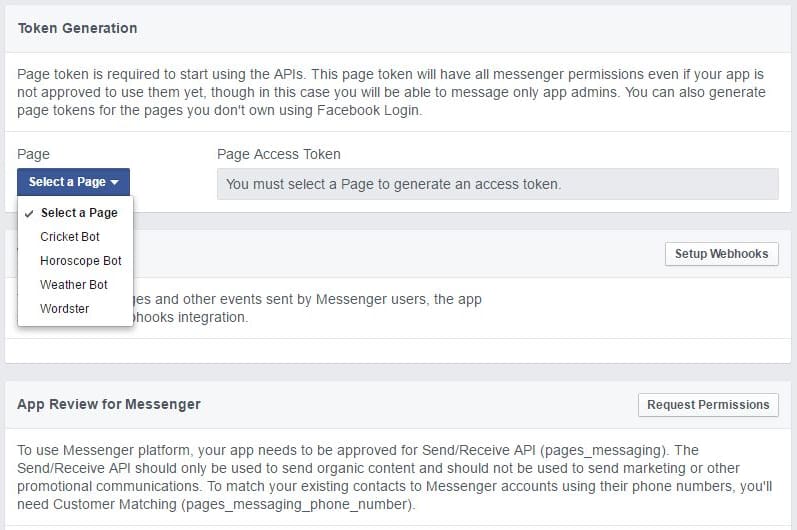
Once created click on Add Product in the left-menu bar and choose Messenger. Go to the Webhooks section and setup your Callback URL and Verify Token. The Callback URL and Verify Token can be found in our documentation 



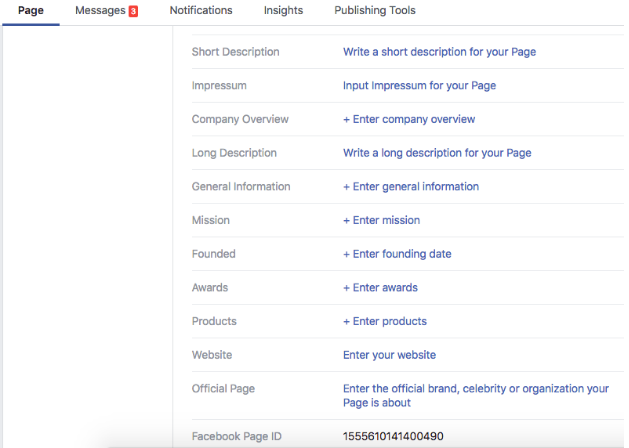
Then [create a Facebook Page](https://www.facebook.com/pages/create) for your app. 



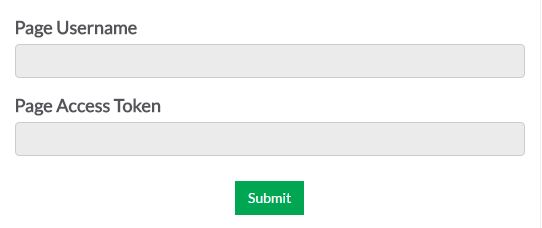
The Page does NOT have to be publicly visible,   
Go back to the app you've just created and find the Token Generation section. Select the Facebook Page you have just created. A Page Access Token will be generated for you. 



You can get your Facebook Page ID by clicking on the 'About' tab on your page and then scrolling right to the bottom. 

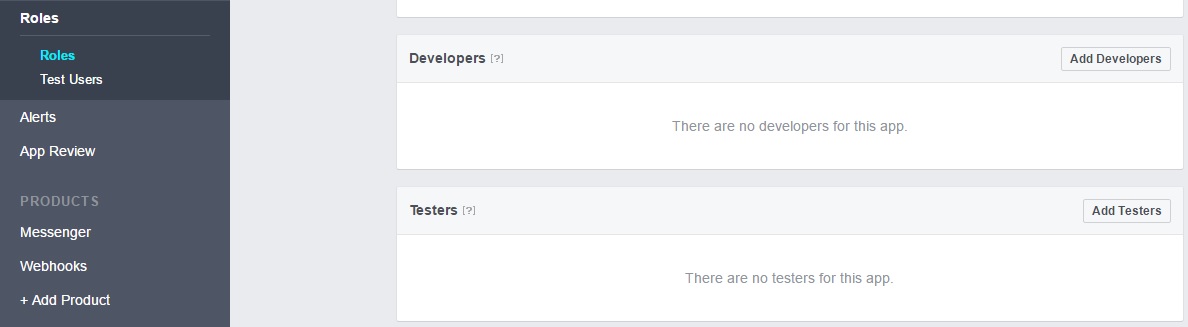


Submit your Page ID, page access token and the welcome message to Gupshup's Publish page and your bot will be live on Facebook. 



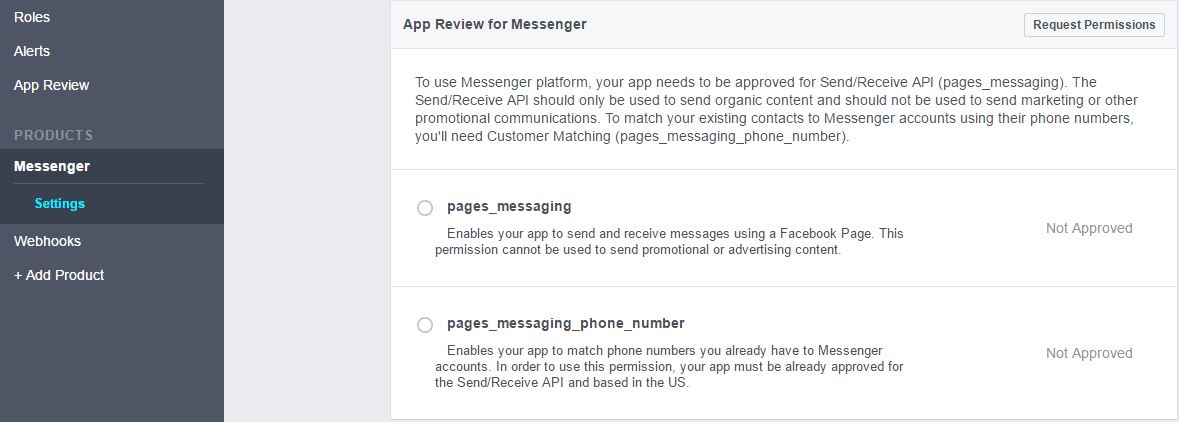
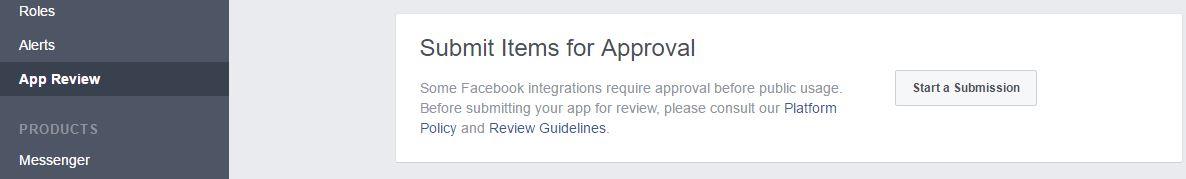
Hit submit and your bot will be live. Your bot will need to be approved/reviewed by Facebook before the public can access it. You can add testers to try a chatbot linked to your Facebook Page while your app is under review.

1.In your App open the 'Roles' section.   
2.Find the 'Testers' group in the bottom of the page and click Add Testers.   
3.Add people to test your app.



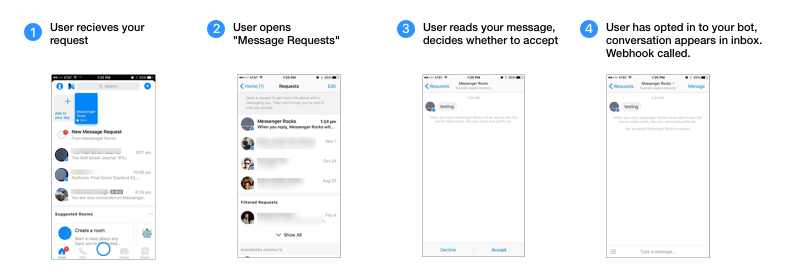
#### 4.   Publish/App Review by Facebook:

**Prerequisite**  
1. Icon - You can upload a JPG, GIF or PNG file. The size of the image must be 1024 x 1024 pixels. File size limit 5 MB  
2. App display name.  
3. App namespace.  
4. Contact Email.  
5. Category.  
6. Privacy policy URL  
7. Terms of Service URL  
8. Video - You'll be prompted to upload a screen cast of your chatbot working.  
9. Facebook page.

In the Messenger section press the 'Request Permissions' button. Choose pages\_messaging from left panel in the popup that appears.   
Fill required fields and upload a video showing your chatbot in action.   
  
In the App Review section click on 'Submit for Review'.   
It will take anywhere between one day to a week until your App is reviewed by Facebook.   
You'll be prompted to upload a screen cast of your chatbot working.   


# Customer matching

Facebook allows you to connect your existing customers to your bot using their phone number.   
Using this feature you can send a messenger request to the customers using their phone number which is mapped to their Facebook account. Once the customer accepts the messenger request your bot receives an event using which you can trigger the user onboarding messages.

   
- image courtesy Facebook documentation

This feature requires you to pay a $99 USD fee to the Facebook and also apply for **pages\_messaging\_phone\_number** permission when submitting your bot for approval. For more details on getting access to this feature please go through this [documentation](https://developers.facebook.com/docs/messenger-platform/guides/customer-matching#access) from Facebook.

### Sending the Messenger request to the customer

While using the Gupshup platform you can utilise this feature by using the API **“api/bot/{botname}/msg”** and sending a specific context to it. We also allow you to send an introductory message along with the messenger request. To access this API go to [API tab](https://www.gupshup.io/developer/ent-apis) –>Exapand “IP Messaging” then find the API.

Here is the curl to use this API -

curl -X POST --header 'Content-Type: application/x-www-form-urlencoded' --header 'Accept: application/json' --header 'apikey: <your API key>' -d 'context=%7B%22botname%22%3A+%22%3Cyour\_bot\_name%3E%22%2C%22channeltype%22%3A+%22fb%22%2C%22contextid%22%3A+%22%3Cphone\_number\_with\_country\_code%3E%22%2C%22contexttype%22%3A+%22phone%22%7D&message=<Into message>' 'https://api.gupshup.io/sm/api/bot/{Your bot name}/msg'

There are two parameters in this curl:

1. **context**: which takes this specific JSON for the feature
2. {"botname": "<your\_bot\_name>","channeltype": "fb","contextid": "<phone\_number\_with\_country\_code>","contexttype": "phone"}
3. **message**: In this you can send the introductory message.

Once you send the message using this API, Facebook will attempt to match the phone number to a user and based on the success and failure you will recieve the response codes back form this API. 

**Expected response codes from this API**:

1. 200 : this means a valid number is sent and matched.
2. 400: Check the message in the response
   1. The response body will have a “code” field with value “10” and message as `Requires phone matching access fee to be paid by this page unless the recipient user is an admin, developer, or tester of the app.`
   2. The response body will have a “code” field with value “100” and message as `No matching user found`.

**NOTE:** This will only work for the bots published on FB channel. If the bot is not yet FB approved then this feature will only work for admins/developers/testers of the Facebook app.

### Handling the user acceptance

Once the user taps on “Accept”, It triggers the [optin](https://www.gupshup.io/developer/docs/bot-platform/guide/messenger_Opt-in_Plugin) event. You will receive a **botoptinevent** in the **EventHandler** of the IDE Bot Builder code.   
You can then choose to start the main flow of the bot.

Sample code:

function EventHandler(context, event) {

if(event.messageobj.text == 'botoptinevent'){

context.sendResponse("Welcome to my store. How can I help?");

}

}