On This Page

Instant Games Quick Start

This tutorial will guide you through building your first Instant Game: a turn-based version of the classic pen and paper game Tic-Tac-Toe. It will use **Rich Gameplay Features** and a **Game bot**.
You can download the source code below:

Download demo source (.zip)

Sections:

Setting up your app

Even if you already have a web game hosted on Facebook.com, or a mobile game on a mobile application storefront, you'll need to **create a new app** for your Instant Game. You cannot have an Instant Game running on the same Facebook App ID as any other app.

Set Category to "Games"

Go to the **Basic Settings** section of your app and make sure to set the category to **Games**. Save your changes.

Add Administrators, Developers and Testers to the App

As Facebook gradually rolls out Instant Games to a wider audience, your game may not be accessible to everyone. To ensure access for all of your developers and testers, add them as Administrators, Developers, or Testers of the Application in the **Roles** tab of the App Dashboard (see screenshot below). Learn more about roles.

Enable Instant Games in the App Dashboard

> **Please Note**: Do not add Instant Games to a pre-existing Facebook App configured for iOS, Android or Web. The Facebook App ID created for your Instant Game cannot be used for another platform.
>
> Create new Facebook App

To enable* Instant Games for the app, click the **+ Add Product** button at the bottom of the navigation menu. This will open a list of products available to the app. Find **Instant Games** on the list and click the **Get Started** button. This will add an **Instant Games** section to the menu. Make sure to enable the the **Use Instant Games** toggle. Save your changes before continuing.

First steps with the SDK

Importing

Now that your app is setup, you need to start creating your game client. The game client needs to have an `index.html` file in the root directory. Start by importing the Instant Games SDK from this file.

```
<script src="https://connect.facebook.net/en_US/fbinstant.6.0.js"></script>
```

Initializing

Our SDK makes extensive use of Promises for asynchronous functionality. You'll only be able to interact with the loading UI after `FBInstant.initializeAsync()` resolves.

```
FBInstant.initializeAsync()
  .then(function() {
    // Start loading game assets here
  });
```

Our game client will not download your bundle (`.zip` file) all at once. Instead it will search the root for configuration (`fbapp-config.json`) and the main file (`index.html`). It will then start executing the logic contained in the main file, and will start downloading assets from there. As a developer you have full control of the order and time in which your assets are loaded.

Once you start downloading the necessary assets for initializing the game, you need to inform the SDK about the loading progress in order for us to display the loading ring to players.

```
var images = ['sprite1', 'sprite2', ...];
for (var i=0; i < images.length; i++) {
  var assetName = images[i];
  var progress = ((i+1)/images.length) * 100;
  game.load.image(assetName);

  // Informs the SDK of loading progress
  FBInstant.setLoadingProgress(progress);
}

// Once all assets are loaded, tells the SDK
// to end loading view and start the game
FBInstant.startGameAsync()
  .then(function() {
    // Retrieving context and player information can only be done
    // once startGameAsync() resolves
    var contextId = FBInstant.context.getID();
    var contextType = FBInstant.context.getType();

    var playerName = FBInstant.player.getName();
    var playerPic = FBInstant.player.getPhoto();
    var playerId = FBInstant.player.getID();

    // Once startGameAsync() resolves it also means the loading view has
    // been removed and the user can see the game viewport

    game.start();
  });
```

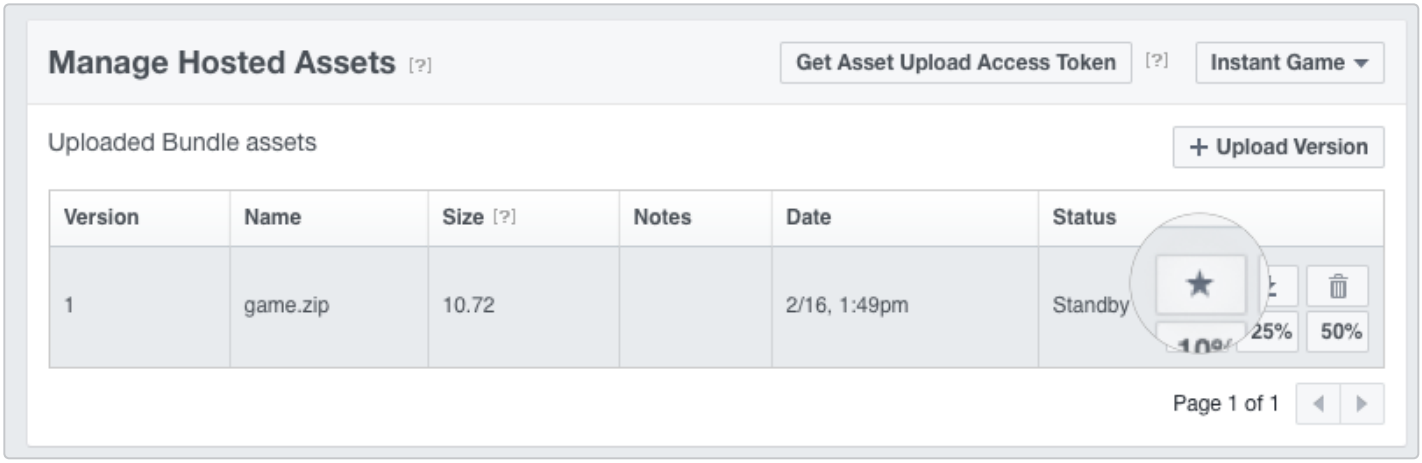
For more information about the `initializeAsync()`, `setLoadingProgress()` and `startGameAsync()` methods, please refer to the SDK Reference.

Testing and Uploading

Instant Games content is hosted on Facebook infrastructure, so you don't need to host the game content on your own or use third-party services. Once the game is ready for testing, package all game files into a single .zip file. Please note that the `index.html` file should be in the root of this archive and not in any sub-folders.

To upload the .zip file, click the **Web Hosting** tab in the App Dashboard. From there select "Instant Game" from the dropdown menu and click **+Upload Version** which will allow you to upload the .zip file to Facebook's hosting service.

After that, the build will process the file, which should only take a few seconds. When the state changes to "Standby", click the **"★"** button to push the build to production.



From this moment on, you can test the build in your mobile device. The published build will now be visible to you in the games list inside of Messenger, under a section caled "In development". To speed up the development process, you can follow this guide on how to upload your build from the command line via the Graph API, or to test directly from your development server: Testing, Publishing and Sharing your Instant Game

Saving State

While developing your game, you'll want to save state across sessions. This enables players to compete, collaborate and to continue playing from where they left off. There's two main ways of storing state: player storage and context storage.

Storing data for a player

You can save player information in a built-in key-value storage provided by the Instant Games SDK. Information stored through these methods will be always available to the player (and to that player only) across all different contexts, and devices for both Messenger and Facebook.

In the case of the Tic-Tac-Toe demo, we are using the player storage methods to store information about the player's "win streak": the amount of matches in a row the player has won. This information will be visible to that player in all contexts that player is playing.

Saving and retrieving data will be available only after `FBInstant.initializeAsync()` resolves.

```
FBInstant.player.setDataAsync({
 'score':currentWinStreak
});
```

```
FBInstant.player.getDataAsync(['score'])
  .then(function(data){
    if (typeof data['score'] !== 'undefined') {
      playerInfo.winStreak = data['score'];
    }
  });
```

Storing data for a context

Depending on the needs of your game you may want to save information that is relevant to that specific context. With Activity Stores you can have key-value storage that all players on that conversation can read and write to. For more information about storing and retrieving context data with Activity Stores, refer to our Activity Stores Guide

Context Updates

Now that we have already covered the basics on working with the Instant Games SDK, let's take a look at how to make your game interact with the context in which it's being played.

We use the word context to define any environment in which a game can be played. More commonly, the context identifies a Messenger conversation, but it can also identify a variety of other things, like a Facebook post or group if the game is not being played in Messenger.

The example below shows how to send a context update and how it will look in a Messenger conversation.

Step 1: Declare templates on a configuration file

In order to declare your custom updates, you need to create a configuration file called `fbapp-config.json` and place it in the root of your bundle, together with your `index.html` file. For more information on the supported configurations, please refer to the Bundle-based configuration section. For the purpose of this demo, the file contents should be as follows:

```
{
  "instant_games": {
    "platform_version": "RICH_GAMEPLAY",
    "custom_update_templates": {
      "play_turn": {
        "example": "Edgar played their move"
      }
    }
  }
}
```

> The custom update template configuration allows us to assign an ID to each specific custom update which results in better analytics. It is mandatory to assign template IDs for all games.

Step 2: Send a custom update with `updateAsync`

Once your template has been declared in your configuration file, you can use to populate the mandatory `template` field in `FBInstant.updateAsync`. Here's how the call is used in Tic-Tac-Toe to communicate the opponent that it's now their turn. To the right you can see what the message will look like.

```
  // This will post a custom update. If the game is played in a messenger
// chat thread, this will post a message into the thread with the specified
// image and text message. And when people launch the game from this
// message, those game sessions will be able to access the specified blob
// of data through FBInstant.getEntryPointData().
FBInstant.updateAsync({
  action: 'CUSTOM',
  cta: 'Play',
  image: base64Picture,
  text: {
    default: 'Edgar played their move',
    localizations: {
      en_US: 'Edgar played their move',
      es_LA: '\u00A1Edgar jug\u00F3 su jugada!'
    }
  }
  template: 'play_turn',
```

```
  data: { myReplayData: '...' },
  strategy: 'IMMEDIATE',
  notification: 'NO_PUSH'
}).then(function() {
  // closes the game after the update is posted.
  FBInstant.quit();
});
```

For more information on custom context updates, please check out our Instant Games SDK Reference.

For a guide on best practices explaining when to message other players, when to notify them and what's the best content to include in these updates, please refer to our Best Practices section.

Note that context updates are not sent outside Messenger. It can be useful to tailor your experience by using the `context.getType()` method, and detecting `THREAD`. You can switch to a more appropriate context using `context.switchAsync`, `context.chooseAsync` or `context.createAsync`.

(Optional) Setting up a Game Bot for re-engagement

An interesting feature of Instant Games is the ability to have a Messenger Platform Bot attached to it. Although this is optional, it gives your game a powerful channel for re-engagement. The guide below shows you how to create and set up your Game Bot.

Step 1: Create a Page

In order to create a Game bot, you'll first need to create a Facebook page. In order for the page to work correctly with your Instant Game it needs some some special properties:

1. The page's category needs to be **App Page**
2. The page's name needs to **contain the name** of the app.
3. The page **cannot be associated** with another app.

You can create a page with these special conditions by going to the **App Page** section of the **Instant Games** product in your App dashboard. Before moving on to the next step, make sure that your **App Page** section looks like the step on the right:

> **Note**: If your Instant Game is not correctly associated with a page as explained above, your bot will not be able to receive `messaging_game_plays` events

Step 2: Activate your Bot

After creating you page, you'll need to make sure to respond to its messaging webhooks. Webhooks are HTTP calls that we send to your backend when a messaging event is sent to your page. Your server's logic will then decide how to properly respond to each event, if a response is appropriate. To associate your server's endpoints with your page events, follow the instructions on the Messenger Platform Quickstart Tutorial to enable the bot for your page. The table below contains information about the webhooks and permissions you will need to make your bot work with Instant Games:

Section	Values
Page events	`messages` and `messaging_game_plays`
Permissions	`pages_messaging`

> Instant Games bots are only permitted to use standard messaging and the `GAME_EVENT` message tag but not `pages_messaging_subscriptions`.
>
> If your bot has other functionality that requires subscription messaging or customer matching you should create a separate app and apply for Messenger platform permissions again.

Step 3: Respond to `messaging_game_plays` webhooks

Once your bot is correctly configured, your server application will start receiving `messaging_game_plays` web hooks every time a player closes the Instant Game. Below is an example of a server application detecting and responding to one these web hooks.

```
if (event.game_play) {
  var senderId = event.sender.id; // Messenger sender id
  var playerId = event.game_play.player_id; // Instant Games player id
  var contextId = event.game_play.context_id;
  var payload = event.game_play.payload;
  var playerWon = payload['playerWon'];
  if (playerWon) {
    sendMessage(
```

```
      senderId,
      contextId,
      'Congratulations on your victory!',
      'Play Again'
    );

  } else {
    sendMessage(
      senderId,
      contextId,
      'Better luck next time!',
      'Rematch!'
    );
  }
}
```

You can refer to the Messenger Platform documentation for more information on this webhook: Game Play Webhook Documentation.

Step 4: Bringing your players back into the game

Below is an example of how to use the Graph API to send a game_play button to your players.

```
curl "https://graph.facebook.com/v2.6/me/messages?access_token=<PAGE_ACCESS_TOKEN>"
   -X POST
   -H "Content-Type: application/json"
   -d '{
  "messaging_type": "UPDATE",
  "recipient": {
    "id": "<RECIPIENT_ID>"
  },
  "message": {
    "attachment": {
      "type": "template",
      "payload": {
        "template_type": "generic",
        "elements": [
          {
            "title": "It has been a while since your last game. Time to get back",
            "buttons": [
              {
                "type": "game_play",
                "title": "Play Tic-Tac-Toe.",
                "payload": "{}",
                "game_metadata": {
```

```
                "context_id": "<CONTEXT_ID>"
              }
            }
          ]
        }
      ]
    }
  }
}'
```

You can refer to the Messenger Platform documentation for more information on this button: Game Play Button Documentation.

Step 5: Follow our guidelines and policies

Before it is launched to production, your game bot should go through Messenger Platform submission process. Make sure to follow the Best Practices below before submitting your bot:

Do:

- Provide **relevant**, **timely** and **valuable** updates to the players. For more information, visit our Best Practices section.
- Give the user control (for example, by confirming they would like to be notified, and with what frequency).
- Use entry point data on play buttons to load the game in contextually relevant ways.
- Name the bot the same as the game.
- Make use of social updates like turn reminders, tournaments results, timed rewards and challenges.
- Make sure your players have the right incentives open the game via a bot message by using the message payload to reward them in-game with something valuable. A bot message is usually not valuable if it opens your game to the start screen.
- Use a persistent menu to provide common actions, such as launching the game.
- Set default action to use `game_play` on custom updates, so that the entire image takes you into the game.
- Use bots to announce new features or content.
- Optimize time of day for message sends per user, being sensitive to timezones.
- Follow the general best practices for Messenger Bots.

Do not:

- ✗ Send a message immediately after the player closes the game.
- ✗ Send messages to re-engage the player with no context (e.g.: "Come back to the game now!"). Instead prefer re-engagement messages with rich context (e.g.: "Your scout has come back with more info")
- ✗ Adopt the voice of other Facebook users or mislead players to believe their friends are communicating with them.
- ✗ Continue to send a user bot messages when they repeatedly do not engage. Policy limits will apply and block your message from being sent. **Current limits are 5 messages over 10 days of last game play session.** More information in section 9.4 of our Platform Policy Docs
- ✗ Link to any app store.

Next steps

Now that you know how to build and configure your **Instant Game** and your **Game bot**, make sure to check the guides below:

- Best Practices - Best practices and tips to optimize your game's performance
- Launch Checklist - Everything you need to check before submitting your game
- FAQ - Frequently Asked Questions and Troubleshooting.

App Center

Facebook Gameroom

Gaming Community

LANGUAGES

English (US) Deutsch Bahasa Indonesia Español Français (France) العربية 中文(简体) Português (Brasil) Italiano 日本語

Products

Facebook Login

Sharing on Facebook

Games

Facebook App Ads

SDKs

iOS SDK

Android SDK

JavaScript SDK

PHP SDK

Unity SDK

Tools

Graph API Explorer

Open Graph Debugger

Object Browser

JavaScript Test Console

API Upgrade Tool

Facebook Analytics

Support

Platform Status

Developers Group

Marketing Partners

Bugs

News

Blog

Success Stories

Videos