Article

Starting and updating Live Activities with ActivityKit push notifications

Use ActivityKit to receive push tokens and to remotely start, update, and end your Live Activity with ActivityKit notifications.

Overview

ActivityKit offers functionality to start, update, and end Live Activities from your app. Additionally, it offers functionality to receive push tokens. Use push tokens to send ActivityKit notifications from your server to Apple Push Notification service (APNs). With notifications, you can:

- Update and end Live Activities
- Start Live Activities
- Update or end Live Activities for a large audience with channels and broadcast push notifications

Related session from WWDC23

Session 10185: Update Live Activities with push notifications

Start and update Live Activities

To start and update your Live Activities by sending ActivityKit push notifications to a device:

1. Use ActivityKit functionality to obtain push tokens.
2. Set up a remote notification server or make changes to your existing server to support ActivityKit push notifications. If you're new to using push notifications, plan to spend time

implementing your remote notification server. For more information, refer to Registering your app with APNs.

3. Send the push tokens you receive from ActivityKit to your server and use them establish a token-based connection to APNs, as described in Establishing a token-based connection to APNs.
4. Handle updates to the push tokens and update or invalidate them on your server as needed.
5. Send ActivityKit push notifications from your server with APNs using the push tokens, HTTP header fields, and JSON payloads described in this article to start, update, and end Live Activities.
6. Test ActivityKit push notifications locally and verify that your Live Activity behaves as expected.
7. Decide the frequency for your ActivityKit push notifications. For more information, refer to the "Determine the update frequency" section below.

To start and update your Live Activities with ActivityKit push notifications on a channel:

1. Enable broadcast capabilities for your app as described in Setting up broadcast push notifications.
2. Set up a remote notification server or make changes to your existing server to support ActivityKit push notifications. If you're new to using push notifications, plan to spend time implementing your remote notification server.
3. Use channel creation request to create a channel for each Live Activity event. For more information, refer to Sending channel management requests to APNs.
4. Send the channel ID for the Live Activity event to your app.
5. When you request a new activity with ActivityKit to subscribe for updates on the channel, use the channel push type with the channel ID.
6. Send ActivityKit push notifications from your server to APNs using the channel ID, HTTP header fields, and JSON payloads described in this article to update and end Live Activities.
7. Test ActivityKit push notifications locally and verify that your Live Activity behaves as expected.

Add the Push Notifications capability

In your Xcode project, start by adding the Push Notifications capability to your app in Xcode as described in Registering your app with APNs. Note that you can't use the User Notifications framework to register your Live Activity for push notifications. Instead, you use ActivityKit to obtain a push token as described below. If you choose to use broadcast push notifications for your app, refer to Setting up broadcast push notifications for more details on enabling broadcast capability for your app.

> **Note**
>
> You can't enable broadcast capability through Xcode. You can only enable broadcast capabilities using developer.apple.com.

Create a push notification server

Additionally, set up a remote notification server or make changes to your existing server and send JSON payloads to APNs as described below. If you're new to using push notifications, plan to spend time implementing the remote notification server. For more information, refer to Registering your app with APNs. To use device push notifications, you use the push tokens that you receive from ActivityKit to send push notifications in your remote notification server. For more information about sending push notifcations using push tokens, refer to Sending notification requests to APNs.

For broadcast push notifications, use the channel you created for the Live Activity and shared with the app to send push notifications in your remote notifications server. For more information on creating channels, refer to Sending channel management requests to APNs. For information on how to send a push notification on a channel, refer to Sending broadcast push notification requests to APNs.

Start a Live Activity that supports push updates to push tokens

On devices that run iOS or iPadOS 17.1 and earlier, you can't start Live Activities with ActivityKit push notifications. However, you can update and end them with ActivityKit push notifications. In your app, start a Live Activity with the `request(attributes:content:pushType:)` function and pass `token` to its `pushType` parameter. As a result, you receive a push token that's unique to your Live Activity when you successfully start the Live Activity.

The following code snippet from the Emoji Rangers: Supporting Live Activities, interactivity, and animations sample code project starts a Live Activity and requests a push token:

```
func startActivity(hero: EmojiRanger) throws {
    let adventure = AdventureAttributes(hero: hero)
    let initialState = AdventureAttributes.ContentState(
        currentHealthLevel: hero.healthLevel,
        eventDescription: "Adventure has begun!"
    )

    let activity = try Activity.request(
        attributes: adventure,
```

```
        content: .init(state: initialState, staleDate: nil),
        pushType: .token
    )

    // ...
}
```

Every time you start a new Live Activity, you need to send the push token to your server. However, the `pushToken` isn't available immediately after calling `request(attributes:content:pushType:)`, because push token creation happens asynchronously and may take some time. Take this into account before accessing the `pushToken` synchronously. Instead, observe push token updates using the `pushTokenUpdates` asynchronous sequence. This makes sure you receive the first push token, and any updates to it.

For example, the following code snippet from the Emoji Rangers: Supporting Live Activities, interactivity, and animations sample code project uses an asynchronous `Task` to receive push token updates. In the asynchronous for loop, it logs the updated token to the console for debugging and local testing, and sends it to the server:

```
Task {
    for await pushToken in activity.pushTokenUpdates {
        let pushTokenString = pushToken.reduce("") {
              $0 + String(format: "%02x", $1)
        }

        Logger().log("New push token: \(pushTokenString)")

        try await self.sendPushToken(hero: hero, pushTokenString: pushTokenString)
    }
}
```

The push token for a Live Activity may change throughout its duration. When your app receives a new token, it receives background runtime to process the updated token. Keep track of the push token for each Live Activity. Additionally, invalidate the previous, now-outdated token on your server when you receive an updated token to successfully send subsequent updates.

Start a Live Activity that supports push updates on a channel

You can update or end Live Activities by an ActivityKit push notification sent on a channel. To create a channel for your Live Activity event, refer to Sending channel management requests to APNs. The channel you create has a unique identifier, called a channel ID, that devices use to

subscribe to Live Activity updates. Your remote server provides the Channel ID to your app before the Live Activity starts. In your app, start a Live Activity with the `request(attributes: content:pushType:)` function and pass channel to its `pushType` parameter with the channel ID. If the channel ID isn't a valid channel, the Live Activity fails to start.

The following code snippet starts a Live Activity and subscribes to receive push notifications on channel ID:

```
func startLiveActivityWithChannel(string channelId, initialState) {
    let activity = try Activity.request(
            attributes: adventure,
            content: .init(state: initialState, staleDate: nil),
            //Input your unique channel ID
            pushType: .channel(channelId)
}
```

Start new Live Activities with ActivityKit push notifications

You can start new Live Activities with ActivityKit push notifications. This option is especially useful for tracking events with Live Activities that occur more than once. For example, if someone starts a Live Activity to track their favorite sports team, the sports app might offer to automatically start a Live Activity for each of the team's games. In the future, the person doesn't have to remember to start the Live Activity from the app. Instead, a Live Activity for tracking the game might automatically start from an ActivityKit push notification you sent with your remote notification server and APNs.

To start Live Activities from ActivityKit push notifications, configure your app to support push notifications. Then obtain the token to for starting a Live Activity:

- Use the `pushToStartTokenUpdates` asynchronous sequence to retrieve a push-to-start token that allows you to start new Live Activities. You don't have to start a Live Activity from your app to receive the push-to-start token.
- Send the push-to-start token to your server and use the `pushToStartTokenUpdates` sequence to receive token updates. Similar to the update token, update it on your server when needed and invalidate the old token.
- On your push notification server, use the push-to-start token to send JSON payloads to APNs that start a new Live Activity.
- When the system receives the ActivityKit push notification on a device, it starts a new Live Activity, wakes up your app, and grants it background runtime to allow you to download assets that the Live Activity needs.

- While the system starts the new Live Activity and wakes up your app, you receive the push token you use for updates. To update and end the Live Activity on devices that aren't running iOS 18 or iPadOS 18, use this update push token as if you obtained it by starting a Live Activity from within your app.
- For devices running iOS 18 and iPadOS 18, you can send different APNs payloads to start a Live Activity to listen to updates on a push token or a channel ID. For more information, refer to "Construct the payload that starts a Live Activity" below for more details. If you are using push tokens to send updates, then the system wakes your app and you'll receive new push tokens to use for updates.

> **Note**
>
> You can't use broadcast push notifications to start a Live Activity.

Construct the ActivityKit remote push notification payload

To successfully start, update, or end a Live Activity with an ActivityKit push notification, send an HTTP request to APNs that conforms to the following requirements:

- Set the value for the `apns-push-type` header field to `liveactivity`.
- Set the `apns-topic` header field using the following format: `<your bundleID>.push-type.liveactivity`.
- Set the value for the `apns-priority` header field to `5` or `10`. For more information, refer to "Determine the update frequency" below.

For more information about request headers, refer to Sending notification requests to APNs.

To successfully update or end a Live Activity with an ActivityKit push notification to a channel, send an HTTP request to APNs that conforms to the following requirements:

- Set the value for the `apns-push-type` header field to `liveactivity`.
- Set the `apns-channel-id` header field to the channel you're using.
- Set the value for the `apns-priority` header field to `5` or `10`. For more information, refer to the "Determine the update frequency" section of this article, below.

For more information about request headers, refer to Sending broadcast push notification requests to APNs.

In your request's `aps` dictionary:

- Set the `timestamp` to the current time in seconds since 1970 to allow the system to always display the most recent ActivityKit push update.

- To start a Live Activity, refer to “Construct the payload that starts a Live Activity” below.
- To update a Live Activity, set the value for the `event` key to `update`.
- To end a Live Activity, set the value for the `event` key to `end`. If you end a Live Activity, include the final content state to make sure the Live Activity displays the latest data after it ends.
- Set the fields for the `content-state` key to match your custom `Activity.ContentState` type to ensure the system can decode the JSON payload and update or end the Live Activity. Additionally, don’t use any custom JSON encoding strategies to encode your data, because the system always decodes JSON payloads for Live Activity updates using its default encoding strategies. Custom encoding strategies will result in update failures.
- To mark a Live Activity as outdated with an update, optionally set the `stale-date`. For more information, refer to the “Mark a Live Activity as outdated by setting a stale date” section below.
- To set a custom dismissal date that tells the system to remove an ended Live Activity from the Lock Screen, refer to the “End the Live Activity with a custom dismissal date” section below.
- To alert a person about a critical Live Activity update, optionally provide an `alert` to light up their device and display the expanded presentation on devices that support the Dynamic Island or a banner on devices that don’t support it.

The following payload updates the Live Activity of the Emoji Rangers: Supporting Live Activities, interactivity, and animations example with the latest information. The content of `content-state` must match the properties of the custom `Activity.ContentState` type you declare in your `ActivityAttributes` implementation. In the example, `content-state` matches the properties of the custom `AdventureAttributes.ContentState` type of the sample code project. Additionally, the example payload includes an alert with a custom sound to let the player know that the hero has been knocked down and requires a healing potion.

```
{
    "aps": {
        "timestamp": 1685952000,
        "event": "update",
        "content-state": {
            "currentHealthLevel": 0.0,
            "eventDescription": "Power Panda has been knocked down!"
        },
        "alert": {
            "title": {
                "loc-key": "%@ is knocked down!",
                "loc-args": ["Power Panda"]
            },
            "body": {
                "loc-key": "Use a potion to heal %@!",
```

```
                "loc-args": ["Power Panda"]
            },
            "sound": "HeroDown.mp4"
        }
    }
}
```

In your alert implementation, consider localizing both strings. For more information on displaying alerts for a remote notification, including localized alert messages and a reference of available keys, refer to Generating a remote notification.

Remember that a person's device may not receive a push notification — for example, if they're in an area without a network connection. Similarly, the system ignores an ActivityKit push notification if it arrives after the Live Activity ended. Both cases can cause a Live Activity to display outdated information. To help reduce the chance of showing outdated information, update your Live Activity from your app in addition to push notifications.

> **Note**
>
> When you send ActivityKit push notifications to a device, the system wakes the widget extension to render the UI of your Live Activity.

Construct the payload that starts a Live Activity

The JSON payload for starting a Live Activity with an ActivityKit push notification is similar to the payload you use for updates. However, it comes with the following requirements:

- Set the value of the `event` field to `start`.
- Include an `alert` in the JSON payload.
- Include the `attributes-type` and `attributes` keys.

By including an alert in your JSON payload, you make sure a person gets alerted about the started Live Activity and avoid unexpectedly surprising them. The following sample payload starts a Live Activity for devices that aren't running iOS 18 and iPadOS 18 or later:

```
{
    "aps": {
        "timestamp": 1234,
        "event": "start",
        "content-state": {
            "currentHealthLevel": 100,
```

```
            "eventDescription": "Adventure has begun!"
        },
        "attributes-type": "AdventureAttributes",
        "attributes": {
            "hero": {
                "name": "Power Panda",
                "avatar": "U+1F43C",
                "healthLevel": 100,
                "heroType": "Forest Dweller",
                "healthRecoveryRatePerHour": 0.25,
                "url": "game:///panda",
                "battleCode": "game:///panda/battle",
                "level": 3,
                "exp": 600,
                "bio": "Power Panda loves eating bamboo shoots and leaves."
            }
        },
        "alert": {
            "title": {
                "loc-key": "%@ is on an adventure!",
                "loc-args": [
                    "Power Panda"
                ]
            },
            "body": {
                "loc-key": "%@ found a sword!",
                "loc-args": [
                    "Power Panda"
                ]
            },
            "sound": "chime.aiff"
        }
    }
}
```

For devices running iOS 18 and iPadOS or later, you can add `input-push-channel` with the appropriate channel ID to start a Live Activity and listen for updates on a channel. After you send this payload, you can send updates on the channel to update a Live Activity.

```
        "input-push-channel": "dHN0LXNyY2gtY2hubA==",
        "attributes-type": "AdventureAttributes",
        "attributes": {
            "currentHealthLevel": 100,
```

```
            "eventDescription": "Adventure has begun!"
        }
```

Create a channel ID before you send this payload to a device and ensure that the channel ID is valid. If you send this payload to a device that doesn't run iOS 18 and later, then the Live Activity will not start.

For devices running iOS 18 and iPadOS 18 or later, you can add `input-push-token: 1` to your payload to start a Live Activity and receive a new push token. After you receive a new push token, you can use it to send updates to a Live Activity.

```
        "input-push-token": 1,
        "attributes-type": "AdventureAttributes",
        "attributes": {
            "currentHealthLevel": 100,
            "eventDescription": "Adventure has begun!"
        }
```

Test Live Activity updates locally

During development, verify your JSON payloads and Live Activity updates locally. Using command-line tools and Simulator or a test device, you can send a JSON payload from the command line to APNs and receive Live Activity updates to verify that your payload updates the Live Activity and the Live Activity behaves as expected.

To locally test Live Activity updates during development:

1. Set up your command line as described in the "Send a Push Notification Using a Token" section of Sending push notifications using command-line tools.
2. Add code that logs the push token to the console when you start a new Live Activity in your app.
3. Run your app in Simulator or on a test device and start a Live Activity.
4. Copy the logged push token from the console and set it as the `$ACTIVITY_PUSH_TOKEN` environment variable in Terminal.
5. Use a `curl` command to send a Live Activity update using APNs. Set the value of the `apns-priority` header field to `10` to deliver the update quickly with high priority.

The following example shows a curl command that the Emoji Rangers: Supporting Live Activities, interactivity, and animations sample code project could use to test a Live Activity update.

```
curl \
 --header "apns-topic: com.example.apple-samplecode.Emoji-Rangers.push-type.liveacti
```

```
--header "apns-push-type: liveactivity" \
--header "apns-priority: 10" \
--header "authorization: bearer $AUTHENTICATION_TOKEN" \
--data '{
    "aps": {
        "timestamp": '$(date +%s)',
        "event": "update",
        "content-state": {
            "currentHealthLevel": 0.941,
            "eventDescription": "Power Panda found a sword!"
        }
    }
}' \
--http2 https://api.sandbox.push.apple.com/3/device/$ACTIVITY_PUSH_TOKEN
```

> **Note**
>
> To test ActivityKit push notifications in Simulator, use a Mac with the Apple T2 Security Chip or a Mac with Apple silicon that runs macOS 13 or later.

To locally test Live Activity updates to a channel during development:

1. Set up your command line as described in the "Send a Push Notification Using a Token" section of Sending push notifications using command-line tools.
2. Create a new channel, and set it as `$CHANNEL_ID` variable in your terminal.
3. Use the newly created channel with ActivityKit when you request a Live Activity.
4. Run your app in Simulator or on a test device and start a Live Activity.
5. Use a `curl` command to send a Live Activity update using APNs. Set the value of the `apns-priority` header field to `10` to deliver the update quickly with high priority.

> **Note**
>
> You can also use Push Notifications Console to send ActivityKit push notifications instead of using terminal. For more information, refer to Testing notifications using the Push Notification Console.

If you encounter failures while sending an ActivityKit push notification:

- Ensure the `curl` command succeeded. An error might indicate an incorrect field for the request, or errors setting up the command-line environment.

- Use the Console app to view device logs and look for errors that could help you understand the issue. Processes in Console that may contain relevant information are `liveactivitiesd`, `apsd`, and `chronod`.

Determine the update frequency

The system allows for a certain budget of ActivityKit push notifications per hour. As with other push notifications you send with APNs, you can set the HTTP header field `apns-priority` for your requests to specify the priority of an ActivityKit push notification:

- If you don't specify the `apns-priority` value, APNs delivers the ActivityKit push notification immediately with the default priority of `10` and counts it toward the notification budget that the system imposes.
- If you exceed the budget, the system may throttle your ActivityKit push notifications.

To avoid throttling, you can send a low-priority ActivityKit push notification that doesn't count toward the budget by setting the HTTP header field `apns-priority` to `5`. Consider this lower priority first before using the priority of `10`. In many cases, choosing a mix of priority `5` and `10` for updates prevents your Live Activity updates from being throttled. For example, the Live Activity of the Emoji Rangers: Supporting Live Activities, interactivity, and animations sample code project could use priority `5` for updates that don't require a person's immediate attention, like when the hero heals by a few points. When the hero requires a healing potion, the example would use a priority of `10` to immediately update the Live Activity and let the user know about the important change.

However, your app may need to update its Live Activity more frequently, causing you to hit the budget limit for ActivityKit push notifications. For example, if your app allows people to track a sports event like a basketball match that requires many updates per minute, you may not be able to update your Live Activity often enough.

To allow for use cases like this, you can enable your Live Activities to receive frequent ActivityKit push notifications:

- If your project includes an `Info.plist` file, add the `NSSupportsLiveActivitiesFrequentUpdates` entry to it, and set its Boolean value to `YES`.
- Alternatively, open the `Info.plist` file as source code, add the key `NSSupportsLiveActivitiesFrequentUpdates`, then set the type to `Boolean` and its value to `YES`. If your project doesn't have an `Info.plist` file, add `NSSupportsLiveActivitiesFrequentUpdates` to the list of custom iOS target properties for your iOS app target and set its value to `YES`.

People can deactivate frequent ActivityKit push notifications for your app in the Settings app. In your code, use `frequentPushesEnabled` to detect whether a person deactivated frequent push notifications and display information in your app that asks them to activate frequent

ActivityKit push notifications again. Additionally, send the value of this setting to your server and adjust the update frequency accordingly.

Similar to other asynchronous sequences ActivityKit provides to receive configuration updates, you can subscribe to changes to the frequent Live Activity push notification setting setting with the `frequentPushEnablementUpdates` stream.

For more information on available HTTPs headers and sending requests to APNs, refer to Sending notification requests to APNs.

End the Live Activity with a custom dismissal date

When you end a Live Activity, by default the Live Activity appears on the Lock Screen for up to four hours after it ends to allow people to glance at their phone to refer to the latest information. To change the time until the system removes a Live Activity from the Lock Screen after it ends, include the `"dismissal-date"` entry in the `"aps"` dictionary of your JSON payload. If you don't include the `"dismissal-date"`, your Live Activity uses the system's default dismissal behavior. To dismiss the Live Activity from the Lock Screen immediately after it ends, provide a date for `"dismissal-date"` that's in the past — for example, `"dismissal-date": 1663177260`. Alternatively, provide a date within a four-hour window to set a custom dismissal date.

Mark a Live Activity as outdated by setting a stale date

In some scenarios — for example, if a person enters an area without network connectivity — you might not be able update the Live Activity with new information, causing it to display outdated data. To provide the best possible user experience and let the person know that the Live Activity displays outdated information, add a timestamp in the optional `stale-date` field to provide the time when the system will consider the Live Activity to be stale. With each update, you can advance this `stale-date`. When you can't update the Live Activity for some time until it becomes stale, the `activityState` changes to `ActivityState.stale` at the specified date. Use the `activityStateUpdates` stream in your app to monitor the activity state and respond to outdated Live Activities that haven't received updates. For example, while a person has network connectivity, a sports app could update the Live Activity with the latest game information and advance the stale date. When it becomes stale, it displays text to indicate that the displayed information is outdated.

Display the most important Live Activity in the Dynamic Island

If your app starts more than one Live Activity, provide a `relevance-score` in your JSON payload to determine the order of your Live Activities on the Lock Screen and which of your Live Activities appears in the Dynamic Island:

- If you don't provide a relevance score or if Live Activities have the same relevance score, the system shows the first Live Activity you started in the Dynamic Island.
- If you use different relevance scores, the system shows the Live Activity with the highest relevance score in the Dynamic Island.

The system expects relative values for the relevance score. Assign a higher value for an important Live Activity content update — for example, a score of `100` — and use lower values for less important Live Activity content updates — for example, `50`.

> **Note**
>
> Keep track of the relevance scores you assign for each ongoing Live Activity so you can change the order of them as needed with each Live Activity update.

The following example payload provides a relevance score of `100` to make sure the Live Activity that receives the update appears in the Dynamic Island and at top of the list on the Lock Screen:

```
{
    "aps": {
        "timestamp": 1685952000,
        "event": "update",
        "relevance-score": 100,
        "content-state": {
            "currentHealthLevel": 0.941,
            "eventDescription": "Power Panda found a sword!"
        }
    }
}
```

See Also

Starting a Live Activity

Displaying live data with Live Activities

Display up-to-date data and offer quick interactions in the Dynamic Island, on the Lock Screen, in CarPlay, and on a paired Mac or Apple Watch.

`class` `Activity`

The object you use to start, update, and end a Live Activity.

{} Emoji Rangers: Supporting Live Activities, interactivity, and animations

Offer Live Activities, controls, animate data updates, and add interactivity to widgets.

`NSSupportsLiveActivities`

A Boolean value that indicates whether an app supports Live Activities.

`NSSupportsLiveActivitiesFrequentUpdates`

A Boolean value that indicates whether an app can update its Live Activities frequently.