
# Access Token Developer Guide

November 6th, 2018

# Introduction

Player access to Vivox resources is controlled via Vivox access tokens (VATs). VATs are a special form of JSON web tokens that are signed by the game server using a token signing key, and then delivered by the client to the Vivox system when the player wishes to perform a privileged operation.

VATs have the following characteristics:

- They can only be used once. Once a token has been used for the privileged operation, it cannot be reused, even for the same operation.
- They expire even if never used. A token may not be used after the expiration time set by the token generation.

Each privileged operation requires a token in a specific format. Privileged operations include:

- Logging in
- Joining a channel (whether muted or unmuted)
- Kicking a user from a channel (only supported in the Server to Server API)
- Muting/Unmuting a user in a channel, for all other users in the channel (only supported in the Server to Server API)

# Getting Started

Before either a client or a server can generate a token, you need an *VivoxIssuer* and *VivoxInsecureSecret/VivoxSecret* from the Vivox Developer Portal.

In order for your game client to be able to access the Vivox system, your game client will need to get VATs for the operations that it wishes to preform. This work can be done in stages. These stages are:

1. Use game client generated tokens.  This is appropriate only for prototyping/testing purposes.  These tokens can be created by the functions included in *FVivoxVoiceChat*.
2. Use secure game server generated tokens. This is necessary for secure deployment of a production game and avoiding token errors due to user time settings.  These can be created by the functions included in *FVivoxToken*.

  

# Generating Tokens on the Client with FVivoxVoiceChat

Important: The following VAT generation methods should only be used during initial development/prototyping. This method of generation should not be included in production builds of your game. Allowing token generation on the client during production is both a security risk and can cause unexpected token expiration errors for your users.

Two functions are provided to generate VATs for the basic operations of login and join channel.

## Login

```
FString FVivoxVoiceChat::InsecureGetLoginToken(const FString&
PlayerName);
```

Parameters:

- *PlayerName*: This should only be the *PlayerName* without the *VivoxIssuer*. For example, if your *PlayerName* is "`0verl0rd`", this *PlayerName* parameter should be "`0verl0rd`".

## Join Channel

```
FString FVivoxVoiceChat::InsecureGetJoinToken(const FString&
ChannelName, EVoiceChatChannelType ChannelType,
TOptional<FVoiceChatChannel3dProperties> Channel3dProperties)
```

Parameters:

- *ChannelName*: This should only be the *ChannelName* without the *VivoxIssuer*. For example, if your *ChannelName* is "`0verl0rd-group-channel`", this *ChannelName* parameter should be "`0verl0rd-group-channel`".
- *ChannelType*: *Non-Positional*, *Positional*, or *Echo*.
- *Channel3dProperties* (optional): *Channel3dProperties* object with settings for the channel, if needed.

Note: This function can only be called when logged in.

  

# Generating Tokens on the Server with FVivoxToken

Before releasing your game to production, you will need to generate your VATs on a secure server and vend those VATs to the game client. Allowing token generation on the client during production is both a security risk and can cause unexpected token expiration errors for your users.

Two functions are provided to generate VATs for the basic operations of login and join channel. Please note that although the parameters might be named the same, their behavior may be different than the versions provided in *FVivoxVoiceChat*.

## Login

```
void FVivoxToken::GenerateClientLoginToken(const FString& PlayerName,
FString& Token)
```

Parameters:

- *PlayerName*: This is an *AccountName* similar to what the *FVivoxVoiceChat::CreateAccountName* function returns. For example, if your *VivoxIssuer* name is "`crux`" and your *PlayerName* is "`0verl0rd`", this *PlayerName* parameter should contain "`.crux.0verl0rd.`"
- *Token*: This will be set to the generated VAT.

## Join Channel

```
void FVivoxToken::GenerateClientJoinToken(const FString& PlayerName,
const FString& RoomId, FString& Token)
```

Parameters:

- PlayerName: This is an *AccountName* similar to what the *FVivoxVoiceChat::CreateAccountName* function returns. For example, if your *VivoxIssuer* name is "crux" and your *PlayerName* is "`0verl0rd`", this *PlayerName* parameter should contain ".crux.0verl0rd."
- *RoomId*: This is a shortened version of the URI returned by *FVivoxVoiceChat::CreateChannelUri*. It should be everything following the "`sip:`"

until the "@". For example, if your channel URI was "`sip:confctl-g-crux.0verl0rd-group-channel@mt1.vivox.com`", your *RoomId* should contain "`confctl-g-crux.0verl0rd-group-channel`".

- *Token*: This will be set to the generated VAT.

# Non-FVivoxToken Token Generation

This section will describe how to generate a VAT without using the provided *FVivoxVoiceChat* and *FVivoxToken* functions. If you will be using the provided functions, you may skip this section.

## Identifiers

### Usernames

Note: A username for a VAT is similar to the *FVivoxVoiceChat::CreateAccountName* function returns.

Usernames must:

- Start and end with a period (.)
- Must contain characters chosen only from letters a-z and A-Z, numbers 0-9, and the following characters: = + - _ . ! ~ ( ) %
- Be no more than 63 bytes in length

Usernames start with a period (.), followed by your issuer, followed by a period (.), followed by the user ID, followed by a period (.).

For example, if your *VivoxIssuer* name is "`crux`", the following is a valid username:

```
.crux.0verl0rd.
```

### Channel Names

Note: The channel name for a VAT is similar to the *RoomId* needed for the *FVivoxToken::GenerateClientJoinToken* function.

The channel name must adhere to the following restrictions:

- Length must not exceed 200 characters.

- Must contain characters chosen only from letters a-z and A-Z, numbers 0-9, and the following characters: = + - _ . ! ~ ( ) %

Channel names must start with a "`confctl`" prefix and channel type, followed by your issuer, followed by a period (.), followed by the channel ID.

For example, if your *VivoxIssuer* name is "`crux`", the following is a valid channel name:

```
confctl-g-crux.0verl0rd-group-channel
```

### *VivoxIssuer* and *VivoxSecret*

The *VivoxIssuer* (issuer) and *VivoxSecret* (key) are provided when you create an application on the Vivox Developer Portal.

## Base64url Encoding

Base64url encoding is Base64 encoding with all trailing '=' removed, '+' changed to '-', and '/' changed to '_'. This is defined in RFC 7515: Section 2.

Here is an implementation in PHP:

```
function base64url_encoded($str)
{
     return rtrim(strtr(base64_encoded($str), '+/', '-_'), '=')
}
```

Here is an implementation in Python:

```
import base64
def base64url_encode(s):

     """Return a base64url-encoded str"""
     return base64.urlsafe_b64encode(s).rstrip('=')
```

For an implementation in C#, please refer to RFC 7515: Appendix C.

# Token Format

## Overview

A VAT is a single-use JSON Web Token (JWT Introduction or RFC 7519) with a limited set of private claims (`f`, `t`, and `vxa`) and public claims (`iss`, `exp`, and `sub`). It has the format "`header.payload.signature`": three base64url-encoded parts separated by dots.

Important: Base64url encoding is not the same as base64 encoding. Please read the description under Base64url Encoding.

## Header

The header in a VAT is an empty JSON object, i.e. {}. The implicit token type is JWT and the MAC algorithm is HMAC SHA-256. That is, it is equivalent in practice to the JWT header `{"'alg":"HS256","typ":"JWT"}`.

## Payload

The payload is a base64url-encoded JSON object containing the claims asserted by the token. The claims must appear in the payload in exactly the order they appear in this table. The payload is claim order dependent. Claims in the incorrect order will cause 'invalid signature' or 'malformed payload' errors to be returned on the API calls.

| Claim | Example Value | Description |
|---|---|---|
| iss | owt | issuer |
| exp | 1451606400 | expiration time |
| vxa | join | Vivox action |
| vxi | 1 | serial number, to guarantee uniqueness within an epoch second |
| sub | sip:.kingfisher.1364.@tla.vivox.com | subject; used in third-party call control |

  

| | | |
|---|---|---|
| f | sip:.baldeagle.1973.@tla.vivox.com | from; the SIP URI of the requestor |
| t | sip:confctl-g-Qe3MHlbSq+kXLw/ZkXvngg==@tla.vivox.com | to; the SIP URI of the channel |

## Signature

The signature is the base64url-encoded HMAC of the first two parts:

```
base64UrlEncode(HMACSHA256(base64UrlEncode(header) + "." +
base64UrlEncode(payload), VivoxSecret))
```

That is, we take the encoded header and encoded payload, join them with a dot, HMAC that with our *VivoxSecret*, and encode the whole thing.

## Supported Values for the vxa Claim

The following table indicates the supported values for the Vivox action (`vxa`) claim.

| vxa claim value | Meaning |
|---|---|
| join | channel join |
| join_muted | channel join listen-only |
| kick | kick user from a channel (only supported in Server to Server API) |
| login | sign in |
| mute | admin mute or unmute user in channel, or all users in a channel (only supported in Server to Server API) |

# Example Tokens

In the following examples, the payload and VAT are displayed with whitespace for clarity; neither the payload nor the access token should contain any whitespace. Fields that are not required for a claim are not included in the payload.

## Example 1 - login token

The token in this example was generated for the user "`baldeagle.1973`" to use to login. The expiration time is midnight new year's day UTC. The *VivoxSecret* is "`secret!`".

### header

```
{}
```

### payload (added whitespace for clarity)

```
{
    "iss":"demo",
    "exp":1451606400,
    "vxa":"login",
    "vxi":10047,
    "f":"sip:.demo.baldeagle.1973.@tla.vivox.com"
}
```

### access token

```
e30.eyJpc3MiOiJkZW1vIiwiZXhwIjoxNDUxNjA2NDAwLCJ2eGEiOiJsb2dpbiIsInZ4a
SI6MTAwNDcsImYiOiJzaXA6LmRlbW8uYmFsZGVhZ2xlLjE5NzMuQHRsYS52aXZveC5jb2
0ifQ.9RsWfYNZyLVl23Dae8yC5Ekr6oDCUt9XZa-y9mgM8vw
```

## Example 2 - join token

The token in this example was generated for the user "`baldeagle.1973`" joining a channel "`sip:confctl-g-demo.Qe3MHlbSq@tla.vivox.com`". The expiration time is midnight new year's day UTC. The *VivoxSecret* is "`secret!`" .

### header

```
{}
```

### payload (added whitespace for clarity)

```
{
    "iss":"demo",
    "exp":1451606400,
    "vxa":"join",
    "vxi":446905,
    "f":"sip:.demo.baldeagle.1973.@tla.vivox.com",
```

```
    "t":"sip:confctl-g-demo.Qe3MHlbSq@tla.vivox.com"
}
```

**access token (251 octets)**

```
e30.eyJpc3MiOiJkZW1vIiwiZXhwIjoxNDUxNjA2NDAwLCJ2eGEiOiJqb2luIiwidnhpI
jo0NDY5MDUsImYiOiJzaXA6LmRlbW8uYmFsZGVhZ2xlLjE5NzMuQHRsYS52aXZveC5jb2
0iLCJ0Ijoic2lwOmNvbmZjdGwtZy1kZW1vLlFlM01IbGJTcUB0bGEudml2b3guY29tIn0
.YUbQ-5FEZEZMfuU8oS3YZ4IK_EF5A8PFsygj-ehf46Y
```

## Example 3 - kick token

This action is only supported in the Server to Server API and does not use the previous user "baldeagle.1973". For server to server only actions, the Admin User ID must be used rather than a signed in user. The token in this example was generated for the Admin User ID "Demo-Admin" to kick user “kingfisher.1364” out of the channel. The expiration time is midnight new year's day UTC. The VivoxSecret is "secret!" .

**header**

```
{}
```

**payload (added whitespace for clarity)**

```
{
    "iss":"demo",
    "exp":1451606400,
    "vxa":"kick",
    "vxi":303167,
    "sub":"sip:.demo.kingfisher.1364.@tla.vivox.com",
    "f":"sip:Demo-Admin@tla.vivox.com",
    "t":"sip:confctl-g-demo.Qe3MHlbSq@tla.vivox.com"
}
```

**access token**

```
e30.eyJpc3MiOiJkZW1vIiwiZXhwIjoxNDUxNjA2NDAwLCJ2eGEiOiJraWNrIiwidnhpI
jozMDMxNjcsInN1YiI6InNpcDouZGVtby5raW5nZmlzaGVyLjEzNjQuQHRsYS52aXZveC
5jb20iLCJmIjoic2lwOkRlbW8tQWRtaW5AdGxhLnZpdm94LmNvbSIsInQiOiJzaXA6Y29
uZmN0bC1nLWRlbW8uUWUzTUhsYlNxQHRsYS52aXZveC5jb20ifQ.jnl0ScLsZrotXb0Wf
```

```
QfNwRTsROON69-Rl4DandAnaLI
```

# Access Token Error Codes

VivoxVoiceChat responses to the calls requiring VATs may return the following error codes:

| Error Code | Meaning |
|---|---|
| 20120 | Access token already used |
| 20121 | Access token expired |
| 20122 | Access token has invalid signature |
| 20123 | Access token payload claims do not match request parameters |
| 20124 | Access token malformed |
| 20125 | Unknown error using access tokens |
| 20126 | Access token mode not enabled |
| 20127 | Access token service unavailable |
| 20128 | Access token issuer does not match claims |