AT&T Developer Best Practices Guide

Guidelines for API Consumption

Version 1.2

June 6, 2018

Developer Delivery Team (DDT)

Legal Disclaimer

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**You**" and "**Your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to You because AT&T believes the Information may be useful to You. The Information is provided to You solely on the basis that You will be responsible for making Your own assessments of the Information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from Your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

Table of Contents

Table of Tables

Table of Figures

1 Introduction

This document outlines some of the key considerations that developers should keep in mind while interfacing with the AT&T API Platform. Developers should ensure that they understand these guidelines and follow them as applicable to their use cases. This document begins with an overview of some of the key concepts associated with the OAuth 2.0 authorization framework and APIs, and then it outlines associated best practices and guidelines.

2 Concepts

The OAuth 2.0 authorization framework provides a safe and secure way to ensure that an app accessing AT&T APIs provides a customer, an application, or both some context as part of an attempt to access protected resources. The following diagrams show details about how the API is consumed by clients, first by acquiring the token, and then by making calls to the API using the acquired token.

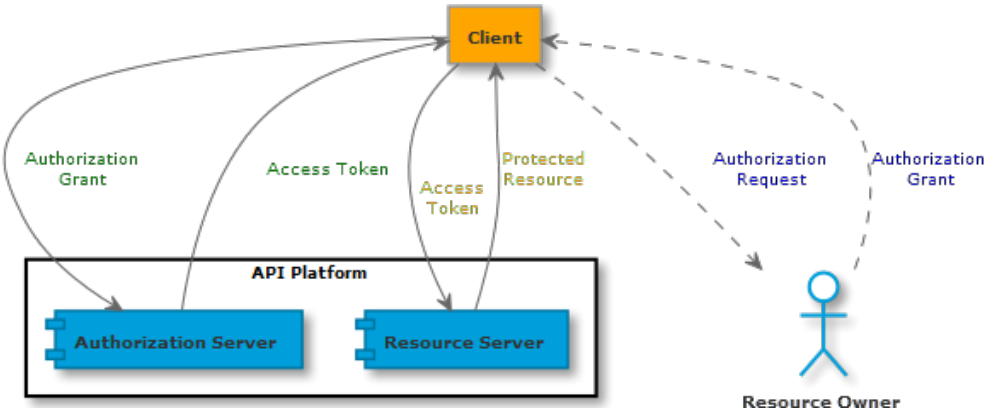

Figure 2-1: OAuth Roles

2.1 OAuth Roles

The OAuth authorization framework has the following roles.

Role	Description
Resource Owner	The resource owner is the entity that owns protected resources.
Client	The client is the entity that accesses protected resources.
API Platform	The API platform is the gatekeeper for the OAuth server and resource server.
Authorization Server	The authorization server is the server that is either the authorization endpoint, or the token endpoint.
Resource Server	The resource server is the server that stores protected resources that the OAuth client accesses.

Role	Description
Access Token	The access token is an identifier to access the protected resources of the resource owner. The OAuth client can use the access token before it expires. The resource server honors the access token until expiry.
Refresh Token	The refresh token is an identifier to obtain access tokens. The authorization server generates the refresh token together with the access token as configured. The OAuth client can use the refresh token to request a new access token from the authorization server when the current access token expires.

Table 2-1: OAuth Roles

2.1.1 Two-legged OAuth Flow

Two-legged OAuth processing requires a grant type of resource owner password credential, or client credentials.

The typical flow for two-legged OAuth processing involves the following activities:

1. An OAuth client initiates a request to an authorization server and receives an access token.
2. The OAuth client uses the access token to access protected resources on the resource server.

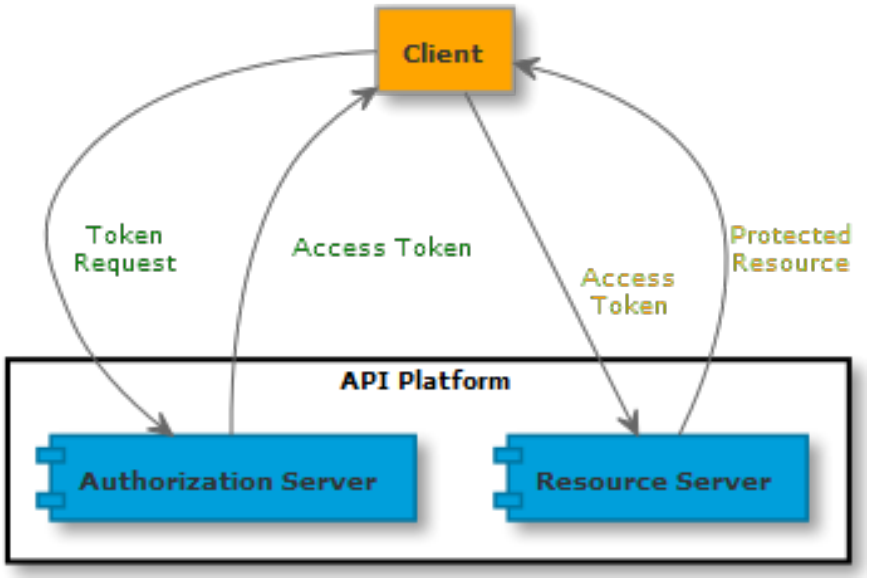

Figure 2-2: Two-legged OAuth Flow

2.1.2 Three-legged OAuth Flow

The typical flow for three-legged OAuth processing involves the following activities:

1. A user, as the resource owner, initiates a request to the OAuth client.
2. The OAuth client sends the resource owner a redirection to the authorization server.
3. The resource owner authenticates and optionally authorizes with the authorization server.
4. The authorization server presents a form to the resource owner to grant access.
5. The resource owner submits the form to allow or to deny access.
6. Based on the response from the resource owner, the following processing occurs:
    a. If the resource owner allows access, the authorization server sends the OAuth client a redirection with the authorization grant code or the access token.
    b. If the resource owner denies access, the request is redirected to the OAuth client, but no grant is provided.
7. The OAuth client sends the following information to the token endpoint (authorization server):
    a. Authorization grant code
    b. Client ID
    c. Client secret or client certificate
8. If verified, the authorization server sends the OAuth client an access token and optionally a refresh token.
9. The OAuth client sends the access token to the resource server to request protected resources.
10. If the access token is valid for the requested resources, the OAuth client can access the protected resources.

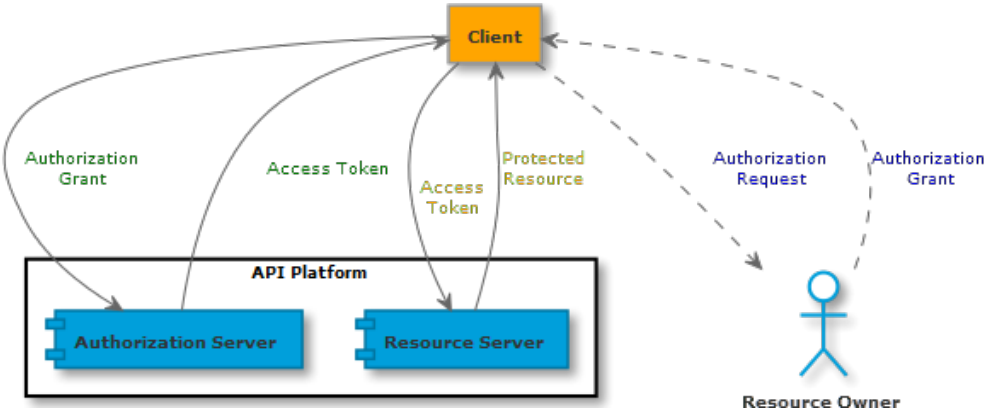

Figure 2-3: Three-legged OAuth Flow

2.1.3 Keywords

The OAuth authorization framework has the following keywords.

Word	Description
Authentication	Authentication to ensure that the entity requesting access to the system is what or who it claims to be
Authorization	Authorization to allow access only to those resources which are appropriate to that entity's identity.
Remember Me	Feature within the AT&T Authentication system, where it remembers the subscriber by dropping the cookie in the browser agent in the subsequent requests.
On-Net Flow	When a customer is on the AT&T Wireless network it is referred to as “on-net-flow." In this flow, the network is able to identify the customer and does not ask for authentication.
Token Context	The following categories classify how the token is acquired. • Token having **resource owner context**: The scope of authorization is under the control of the resource owner. User tokens are typically acquired when user context is required; for example, for managing a user inbox we need to first take permission from the user, and using the token resource server should be able to identify the user belonging to this token. • Token when authorization scope is limited to the protected resources under the direct control of the developer application: This type of token is typically used for making server-server calls without end user permissions.

Table 2-2: OAuth Keywords

2.2 Transient Fault Handling

An application that communicates with remote services must be sensitive to transient faults. This is especially true for the application that is communicating over the Internet like API clients to the API gateway. Transient faults include momentary loss of network connectivity, temporary unavailability of the service, or timeouts that can arise when service is busy. These faults are typically self-correcting and if the action is repeated after a suitable delay, it is likely to succeed.

However, implementing a client application that can handle transient faults for all foreseeable circumstances is difficult. To ensure that applications operate reliably, they must be able to respond to the following challenges:

- The application must be able to detect faults when they occur and determine if these faults are likely to be transient, more long-lasting, or are terminal failures.
- The application developer should devise an appropriate strategy for handling transient faults.

2.3 Errors

Good error handling techniques is one of the basic building blocks of robust and well-behaved applications. The AT&T APIs use standard HTTP status codes as a primary error mechanism. The status code is returned in the response message, in the first line of the message. The first line of the response message is referred to as the status line. It contains the HTTP version, status code, and reason phrase. The rest of the message consists of headers and the message body as shown in the following example.

```
HTTP/1.1 201 OK
Date: Thu, 24 Oct 2016 02:51:59 GMT
Content-Type: application/json
Content-Length: 104
Location: http://api.att.com/musicLibrary/v1/artists/3

{
    "artist": {
        "name": "Aerosmith",
        "genre": "rock",
        "artistId": 3
    }
}
```

Example 2-1: Response with Status Codes, Headers, and Message Body

The status code is a three-digit number starting at 100 and going up to 599 (although a majority of that range is not used). HTTP conveniently groups the status codes together, based on the type of status. Each type of status is grouped by the first digit of the status code. So, by just examining the first digit, you can get an idea of the type of status and in some cases, make a decision based on that.

The HTTP Status Codes have the following groups and descriptions:

Group	Category	Description
1xx	Informational	This group is for status codes that provide informational statuses.
2xx	Successful	The request was understood and accepted.
3xx	Redirection	Further action is needed before the request can be completed.
4xx	Client Error	The request was constructed incorrectly (bad syntax, wrong permissions, etc.) by the client.
5xx	Server Error	The request was valid, but the server was unable to fulfill the request.

Table 2-3: HTTP Status Codes Groups and Descriptions

The following example shows HTTP Status Code 400 Bad Request from the AT&T API Gateway.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 115
Date: Thu, 04 Jun 2014 02:51:59 GMT
x-att-errormessageId: SVC0002
x-att-errortext: Invalid input value for message part %1
x-att-errorvariables: count
x-att-errorType: service
x-att-errorInfo: http://developer.att.com/apis/error-detail?error_code=SVC0002

{
"requestError": {
"serviceException": {
"messageId": "SVC0002",
"text": "Invalid input value for message part %1",
"variables": "count"
}
}
}
```

Example 2-2: HTTP Status Code 400 Bad Request from AT&T API Gateway

The errors are returned in headers and in the body. AT&T errors are heavily constructed from the Open Mobile Alliance (OMA) common error specification. The root of the error is the requestError object. The requestError object defines two types of errors:

- serviceException: Service errors are results of some error that occurred during the running of the service (these errors start by SVCXXXX).
- policyException: Policy errors represent some kind of authorization, or a business error that occurred, which tends to suggest that there is no issue with how the request is formed and sent by the client, but the server does not fulfill the request due to some policy in place (these errors start with POLXXXX).

3 Best Practices

The Best Practices for the developer are reinforced by the Best Practice tests, and are organized in categories.

3.1 Token Reuse

Successful token calls result in the following response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Date: Wed, 30 Mar 2011 07:18:40 GMT

{
"access_token": "b305a68014c81ddd6e9dcf172d2a1064",
"expires_in": 172800,
"refresh_token": "d6439f42285acc03566709c4eea4fd49bc189b72",
"token_type": "bearer"
}
```

Example 3-1: Response for a successful token call

The response contains **access_token** and **refresh_token** pairs along with **expires_in** and **token_type** parameters. The access token parameter is valid until it reaches the value for the **expires_in** parameter, expressed in seconds. In this example, the **expires_in** value is configured to 172800 seconds, which is 48 hours or two days. The refresh token has its own expiry date, which is *not* returned as a response parameter and will be communicated via developer documentation. The refresh token is currently configured to expire in 90 days.

If you are a developer receiving the pair of tokens, you *must* do the following:

- Cache these tokens for making future API calls.
- Reuse the token until it is expired.

When a token is expired, you *must* use the refresh token to acquire the new token pair and so forth.

If you are using the **client credentials** token type, you should *not* acquire the new token until the current token is expired. In a nutshell, under the current token expiry settings, only one token should be enough for making all the calls for next 48 hours, once the new token is acquired.

If you are using the **implicit grant_type**, only the access token is returned. You should cache this token on the client side or the developer-hosted server (DHS) for making further API calls.

3.2 Expired Tokens

The two complementary strategies that developers may consider when handling expired tokens are as follows:

- Track the creation of the OAuth access token and use the refresh token at appropriate intervals based on the OAuth access token creation time, in order to generate a new OAuth access token, before the **expires_in** parameter value for the current OAuth access token has elapsed.

- Capture the response with HTTP Status Code 401- Unauthorized and invoke logic that retries the appropriate Get Access Token method request, as follows:
    - Generate a new OAuth access token.
    - Use the new OAuth access token to resubmit any previously failed method requests that failed due to an expired OAuth access token.

In the future, AT&T may change the default values of the expiration parameters for the OAuth access token and refresh token. So it is strongly advised that you always check the **expires_in** parameter value that is returned with the response to the Get Access Token method call.

3.3 Retries

Some developers build their application with retries, in such cases much care should be taken when coding for retry failures to avoid significant performance implications. If the server or the network has a temporary blip, it might result in heavy retry traffic and can degrade the server performance.

- The **developer should not code retry logic for errors other than the 503 error code.**
- When using **authorization code** or **implicit grant type** do not implement retries, design your flows in such a way that if a retry is required, it is triggered by the end user.
- The developer should not have more than 1-3 retries, and each try should be exponential in terms of back off:
    - **Exponential back-off** - the application waits a short time before the first retry, and then exponentially increasing times between each subsequent retry. For example, it may retry the operation after 5 seconds, 15 seconds, 30 seconds, and so on.
- Retry mechanism between different layers of your application. Avoid logic that includes cascading retry logic at every stage of operation that involves a hierarchy of requests. Use policies that avoid excessive retries. Never implement endless retries. One common way to handle retry in your application is to implement a **circuit breaker pattern**.
- Never perform an immediate retry more than once.
- Avoid using a regular retry interval, especially when you have a large number of retry attempts.
- Prevent multiple instances of the same client, or multiple instances of different clients, from sending retries at the same time.
- Test your retry strategy and implementation, refer to section 4 Application Review.

3.4 Application Credentials

The following precautions must be observed in the handling of the Application Secret for your application in the production realm:

- The Application Secret must only be distributed to authorized and trusted personnel.
- The Application Secret must be stored on a secure server that is set up as follows:
    - Free from computer viruses and unauthorized software.
    - Only accessible by authorized personnel and software.
- The Application Secret is only intended to be used in a server-to-server API request over HTTPS, such as the Get Access Token method, and must never be transmitted or shared with applications on a user's mobile device.
- Application keys and application secrets must not be logged in your system or application logs.
- If unauthorized access to the Application Secret is detected, then the Application Secret for your application must be changed immediately by creating a fresh application on the developer portal. Your application credentials (App key & secret) must be stored on your Developer Hosted Server (DHS) – see section 3.6

3.5 Message Correlation

The developer can use an OAuth redirect URI extension mechanism to implement request response correlation; this can help the developer to maintain the relationship between the token and the original request, for example:

S.No	Registered Application redirect_uri	Overloaded application redirect_uri
1	https://server.com/myuri.html	https://server.com/myuri.html?**param1=value1**
2	https://server.com/	https://server.com/**mydir**
3	https://server.com/	https://server.com/**mydir?param1=value1**

Table 3-1: OAuth redirect URI extension mechanism to implement request response correlation

The following are some of the rules that apply to the redirect URI overloading for maintaining the state information:

- The Fully Qualified Domain Name (FQDN) element of the URI must match the FQDN of the URI provided by the developer at the time of app account creation and subsequent modification on the Developer Program website. This is done by placing a value in the OAuth Redirect URL field, a required field.

- If no value is passed for the parameter, the value provisioned in the app account on the Developer Program website for the OAuth Redirect URL field will be used.
- At run-time, the developer can add additional query parameters to the provisioned OAuth Redirect URI by placing them on the URI string. This allows the app to maintain some state information between initiation of the Get User Authorization method request and the final redirect back to the developer's server. Any query parameters passed in are not used in validation (comparison) between the run-time redirect_uri value and the value of OAuth Redirect URI in the app provisioning details.

3.6 Developer Hosted Server

A developer hosted server (DHS) must be used in order to protect your application keys and application secrets. Optionally, a DHS can be used to:

- Analyze your application traffic to gain more insight about your clients.
- Make your application more resilient by implementing traffic queues when there is a sudden traffic surge.

3.7 Traffic Patterns

Designing an API while keeping in mind the traffic pattern of the application is one of the key aspects of building and running successful APIs.

- You must provide your app traffic projection during the API design phase. What is the expected average traffic that your application will have?
- If your app is seasonal, you must provide those details to AT&T support and infrastructure in advance.
- You must handle traffic bursts using common queuing patterns on your developer hosted server.

3.8 Names, Data Types and Sizes

API documentation specifies the names, data types and sizes for various parameters used in that API.

- Parameter Names must be treated as case sensitive. Even if in practice you observe parmeter names are not being treated in a case sensitive manner, AT&T may change to a stricter policy while validating parameters at any time.
- Data types are to be treated as strict. Even if in practice you see other data types being accepted, AT&T may change to a stricter policy while validating parameters at any time.
- Parameter Sizes must be treated as strict. If no size is specified, industry conventions based on the type of param must be applied. For example, a String parameter in the body can be as long as is supported by the body format type. Even if in practice you see certain sizes being returned, AT&T

may change to a different size at any time so long as it matches industry conventions.

3.9 Error Handling

Errors can be handled in different ways depending on the type as indicated by the HTTP Status Code.

3.9.1 Client Error (4XX)

Many times these errors can be handled during application design time, and if coding is done correctly, we might not see many of these errors during the run time.

The following table lists some of the common errors, and notes about these errors, for example; the message SVC0002 represents that the client did not pass a valid value for a given parameter. The parameter name is sent back as the value of variables parameter.

messageId	Status Code	Notes
SVC0001	400	When an unexpected error arises, this error code is sent back to the client. This is similar to the catch all handler for client errors.
SVC0002	400	This error is thrown when a client makes a mistake while sending a specific parameter.
SVC0003	400	This error is sent back to the client when the server expected enum value, and the client does not send the correct value. The server can send back the applicable valid values to the client in the variables parameter.
SVC1002	400	This error is thrown when a required parameter is missing in the request.
SVC1003	400	This error represents that a parameter is passed in the request which is not required.
SVC1002	404	This error represents that no resource is found referenced by current request URI.
POL0001	429	This indicates the client is sending too many requests, more than the quota allocated to the client. Please check your developer documentation for TPS (Transactions Per Second) allowed for your API.

messageId	Status Code	Notes
POL0001	401	This error indicates that the token parameter is either wrong, or it is expired.
POLXXXX	403	This error indicates the server refuses to fulfill the request due to some policy in place. For example, for a specific API POL9200 indicates that the subscriber is no longer ACTIVE.

Table 3-2: Common Client Errors (HTTP Status Code 4XX)

3.9.2 Request Timeout Error (408)

When you see this error, the retry logic can be implemented as per the guidelines in section 3.3 Retries.

3.9.3 Server Error (5XX)

Error codes like HTTP Status Code 500 Internal server error suggest there is something wrong with the internal state of the server. Repeating this request is not going to help either. The following table lists some of the common 5XX errors.

messageId	Status Code	Notes
SVC0001	500	When an unexpected error happens, this error code is sent back to the client. This is similar to the catch all handler for server errors.
SVC1004	503	This indicates that client traffic has a sudden spike. Client must send requests at a slower rate. **Note**: 503 is the only error code for which the client should enable retry logic, if documented in the API documentation.
SVCXXXX	500	This indicates API specific internal server error, more insight can be provided by sending a new message in response.

Table 3-3: Common Server Errors (HTTP Status Code 5XX)

3.9.4 Redirection (3XX)

AT&T OAuth uses HTTP Status Code 302 heavily to dispense authorization code to the application, as well as implicit user authentication of clients, when they are

using their devices on the AT&T radio network, so the client software should be able to handle 302 traffic.

4 Application Review

The AT&T design and production support teams have a wealth of experience with building and supporting many different kinds of applications, they can provide valuable feedback before the application is released in the production environment.

- It is highly recommended that your application is reviewed by API design and production support teams before you deploy your code in the production environment (this is a mandatory requirement for AT&T branded apps).

5 Appendix: References and Revisions

The following table lists references related to this document.

S.No	Name	Description
1	https://martinfowler.com/bliki/CircuitBreaker.html	Circuit Breaker

Table 5-1: References

The following table lists the revision history for this document.

Date	Revision	Description
9/15/2017	1.0	Version 1
4/09/2018	1.1	Updated section for handling appKey / Secret
6/6/2018	1.2	Added section 3.8 about best practices for names, data types and sizes

Table 5-2: Revision History