Client Credentials Grant

Overview

The Client Credentials Grant is a single-step OAuth 2.0 grant type designed exclusively for non-user applications and service-to-service authentication. Unlike the Authorization Code Grant which requires user interaction, Client Credentials enables applications to authenticate directly using their own credentials without any user context.

Use Cases

Perfect For:

Background Jobs & Scheduled Tasks:
├─ Nightly batch processing
├─ Cron jobs
├─ Scheduled reports
├─ Database cleanup tasks
└─ No user interaction needed

Service-to-Service Integration:
├─ Backend service to Genesys Cloud API
├─ Microservice communication
├─ Application-to-application
├─ System integrations
└─ Automated workflows

Bots & Virtual Agents:
├─ Chatbot integrations
├─ Virtual agent frameworks
├─ Automation engines
├─ API consumers without users

└─ Standalone applications

Data Synchronization:
├─ Salesforce ↔ Genesys sync
├─ Contact import/export
├─ Bulk data operations
├─ Third-party CRM integration
└─ ETL (Extract-Transform-Load) processes

NOT Suitable For:

User-Initiated Access:
├─ Web applications with logins
├─ Mobile apps with users
├─ Interactive scenarios
├─ User-specific data access
└─ Use Authorization Code Grant instead

User Context Required:
├─ When you need to know which user
├─ User-specific APIs (/v2/users/me)
├─ Personalized interactions
├─ Individual user permissions
└─ Cannot use Client Credentials

Single-Step Authentication Flow

Unlike Authorization Code's two-step process, Client Credentials is direct and immediate:

Client Credentials Flow:

Step 1: Direct Exchange
Your App → Sends credentials → Genesys Authorization Server
Step 2: Immediate Response
Genesys Authorization Server → Returns access token → Your App
Step 3: Use Token
Your App → Uses token → Calls Genesys APIs

No User Involved:
├─ No browser redirect
├─ No user login screen
├─ No permission consent
├─ No user interaction
└─ Application acts as itself

Timeline:
├─ Entire flow: milliseconds
├─ No waiting for user
├─ Can happen anytime
├─ No session required
└─ Fully automated

Contrast with Authorization Code:
├─ Authorization Code: 3 round trips (browser, user, server)
├─ Client Credentials: 1 round trip (direct)
├─ Authorization Code: Minutes (user delays)
├─ Client Credentials: Milliseconds
└─ Different use cases, different timing

Complete Client Credentials Flow

Step 1: Application Authenticates

Your application makes direct request to Genesys:

POST https://login.mypurecloud.com/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET
&scope=conversations:readonly+users:manage+scheduling:manage

Request Parameters:

grant_type (required):
├─ Must be "client_credentials"
├─ Identifies this grant type
└─ Tells server how to authenticate

client_id (required):
├─ Your application's public identifier
├─ Identifies which app is requesting
└─ Visible in logs and audit trails

client_secret (required):
├─ Your application's secret credential
├─ NEVER expose in frontend code
├─ Server-side only
├─ Used to prove application identity
└─ Should be rotated monthly

scope (required):
├─ Space-separated list of permissions
├─ Defines what API access is granted
├─ Examples: "users:manage" "scheduling:manage"
├─ Request MINIMUM scopes needed
└─ Note: Cannot exceed application's role permissions

Security Note:
├─ This is direct server-to-server
├─ Both client_id and client_secret sent
├─ client_secret proves application identity
├─ HTTPS encryption required
├─ No user credentials involved
└─ No credentials transmitted to third parties

Step 2: Receive Access Token

Genesys Cloud responds immediately:

HTTP 200 OK

Content-Type: application/json

```
{
  "access_token": "abc123xyz789...",
  "token_type": "bearer",
  "expires_in": 86400,
  "scope": "conversations:readonly users:manage scheduling:manage"
}
```

Response Fields:

access_token:

├─ The access token

├─ Use with all API requests

├─ Include in Authorization header

├─ Proves application authenticated

├─ Example: "Bearer abc123xyz789..."

└─ Expires automatically (default 1 hour)

token_type:

├─ Always "bearer" for Genesys Cloud

├─ Indicates HTTP Bearer token format

└─ Use in header: "Authorization: Bearer {token}"

expires_in:

├─ Token lifetime in seconds

├─ Default: 3600 (1 hour)

├─ Configurable when creating OAuth client

├─ Range: 300 to 172,800 seconds

├─ SCIM special: Up to 450 days

└─ Application must refresh after expiration

scope (informational):

├─ Actual scopes granted

├─ Should match what you requested

├─ Confirms which APIs you can access

└─ Useful for debugging

No Refresh Token:
├─ Client Credentials does NOT include refresh token
├─ Simply authenticate again when token expires
├─ Each authentication is independent
├─ Same as getting fresh token
└─ Simpler than managing refresh tokens

Error Response:

If authentication fails:

HTTP 400 Bad Request
{
  "error": "invalid_client",
  "error_description": "client_id or client_secret is invalid"
}

OR:

HTTP 403 Forbidden
{
  "error": "invalid_scope",
  "error_description": "Application does not have access to this scope"
}

Common Error Codes:
├─ invalid_client: Credentials invalid
├─ invalid_scope: Requested scope not allowed
├─ unsupported_grant_type: Wrong grant type
└─ server_error: Genesys temporary issue (retry)

Step 3: Use Access Token

Your application now has access token

Use to call Genesys Cloud APIs:

GET /api/v2/users

Host: api.mypurecloud.com

Authorization: Bearer abc123xyz789...

Content-Type: application/json

Parameters:

?pageSize=100&pageNumber=1

Example Response:

HTTP 200 OK


```
{
  "entities": [
    {
      "id": "user-123",
      "email": "john@example.com",
      "name": "John Doe"
    },
    ...
  ],
  "pageNumber": 1,
  "pageSize": 100,
  "total": 5000
}
```


Token Usage Rules:

├─ Include in Authorization header

├─ Format: "Bearer {access_token}"

├─ Send with every API request

├─ HTTPS connection required

├─ No other authentication needed

├─ Genesys validates on each request

└─ Application identity verified by token

Important Limitation:

├─ NO user context

├─ Cannot access /v2/users/me (no "me")

├─ Cannot know which user made request

├─ Application acts as itself

├─ No user-specific data access

├─ All requests attributed to app, not user

└─ Perfect for background jobs

Step 4: Token Expires & Refresh

After 1 hour (or configured duration):

Access token expires automatically

Your app's next API request with expired token:

GET /api/v2/conversations
Authorization: Bearer abc123xyz789... (expired)

Genesys Cloud responds:

HTTP 401 Unauthorized
{
  "error": "invalid_token",
  "error_description": "Access token expired"
}

How to Handle Token Expiration:

Option 1: Proactive Refresh (Recommended)
├─ Monitor token expiration time
├─ Refresh 5 minutes BEFORE expiration
├─ Always have fresh token available
├─ No 401 errors encountered
└─ Smoother operation

Option 2: Reactive Refresh
├─ Continue until 401 error
├─ Detect 401 response
├─ Authenticate again (Step 1)
├─ Get new access token
└─ Retry original request

Getting New Token:

Simply authenticate again:

POST https://login.mypurecloud.com/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET
&scope=conversations:readonly+users:manage

Response:
HTTP 200 OK
{
  "access_token": "new_token_def456...",
  "token_type": "bearer",
  "expires_in": 86400,
  "scope": "..."
}

Use new token immediately:
├─ Discard old token
├─ Use new token for requests
├─ Repeat cycle on next expiration
└─ No special refresh_token needed

Token Duration Configuration

When Creating OAuth Client:

Standard Duration:
├─ Default: 86,400 seconds (1 day)
├─ Configurable range: 300-172,800 seconds
├─ Maximum standard: 48 hours
└─ Sufficient for most applications

SCIM Integration Duration (Special):
├─ Up to 38,880,000 seconds (450 days!)

├─ Requires SCIM Integration role
├─ Extended duration for identity provisioning
├─ Reduces authentication frequency
└─ Requires different configuration

HIPAA Compliance:
├─ Overrides token duration
├─ Enforces 15-minute idle timeout
├─ Applies to all OAuth tokens
├─ Security requirement
└─ Automatic when HIPAA enabled

Choosing Duration:

Short Duration (5-60 minutes):
├─ More secure (smaller compromise window)
├─ More frequent authentication needed
├─ Suitable for: High-security, continuous applications
└─ Example: Real-time event processing

Medium Duration (1-8 hours):
├─ Balanced approach
├─ Typical for most applications
├─ Suitable for: Background jobs, daily processes
└─ Example: Hourly data sync

Long Duration (1-2 days):
├─ Fewer authentications
├─ Less secure (larger compromise window)
├─ Suitable for: Long-running background tasks
└─ Example: Large data migrations

Very Long Duration (SCIM only):
├─ 450 days maximum
├─ Minimal authentication needed
├─ Identity provisioning specific
├─ Reduced overhead for identity sync
└─ Requires SCIM Integration role

Recommendation:

```
├─ Start with 1 hour (3600 seconds)
├─ Adjust based on actual usage
├─ Monitor authentication frequency
├─ Balance security vs convenience
└─ Document your choice
```

Complete Python Example

```
import requests
import json
from datetime import datetime, timedelta

class GenesysCloudAPI:
    def __init__(self, client_id, client_secret, region='mypurecloud.com'):
        self.client_id = client_id
        self.client_secret = client_secret
        self.region = region
        self.auth_url = f'https://login.{region}/oauth/token'
        self.api_url = f'https://api.{region}/api/v2'

        self.access_token = None
        self.token_expires_at = None

    def authenticate(self, scopes='conversations:readonly users:readonly'):
        """Step 1: Authenticate using Client Credentials"""

        data = {
            'grant_type': 'client_credentials',
            'client_id': self.client_id,
            'client_secret': self.client_secret,
            'scope': scopes
        }

        try:
            response = requests.post(self.auth_url, data=data)
            response.raise_for_status()
```

```
            # Step 2: Receive token
            token_data = response.json()
            self.access_token = token_data['access_token']

            # Calculate expiration (expire 5 minutes early)
            expires_in = token_data['expires_in']
            self.token_expires_at = datetime.now() + timedelta(seconds=expires_in - 300)

            print(f'[{datetime.now().strftime("%H:%M:%S")}] Authenticated successfully')
            print(f'Token expires at: {self.token_expires_at.strftime("%H:%M:%S")}')
            return True

        except requests.exceptions.RequestException as e:
            print(f'Authentication failed: {e}')
            return False

    def ensure_token(self, scopes='conversations:readonly users:readonly'):
        """Proactive token refresh if expiring soon"""

        if self.access_token is None:
            return self.authenticate(scopes)

        # Check if token expiring soon (proactive refresh)
        if datetime.now() >= self.token_expires_at:
            print('Token expired, refreshing...')
            return self.authenticate(scopes)

        return True

    def get_conversations(self):
        """Step 3: Use token to call API"""

        if not self.ensure_token():
            print('Failed to obtain access token')
            return None

        headers = {
            'Authorization': f'Bearer {self.access_token}',
            'Content-Type': 'application/json'
        }
```

```
    try:
        response = requests.get(
            f'{self.api_url}/conversations',
            headers=headers,
            params={'pageSize': 100, 'pageNumber': 1}
        )
        response.raise_for_status()

        data = response.json()
        print(f'Retrieved {len(data.get("entities", []))} conversations')
        return data

    except requests.exceptions.HTTPError as e:
        if e.response.status_code == 401:
            print('Token expired, retrying with fresh token...')
            if self.authenticate():
                return self.get_conversations()  # Retry
        print(f'API call failed: {e}')
        return None

def create_user(self, email, name):
    """Example: Create new user"""

    if not self.ensure_token('users:manage'):
        return None

    headers = {
        'Authorization': f'Bearer {self.access_token}',
        'Content-Type': 'application/json'
    }

    data = {
        'email': email,
        'name': name,
        'state': 'active'
    }

    try:
        response = requests.post(
```

```
                f'{self.api_url}/users',
                headers=headers,
                json=data
            )
            response.raise_for_status()

            user = response.json()
            print(f'Created user: {user["name"]} ({user["id"]})')
            return user

        except requests.exceptions.RequestException as e:
            print(f'Failed to create user: {e}')
            return None

# Usage Example:
if __name__ == '__main__':
    # Initialize with credentials
    client = GenesysCloudAPI(
        client_id='YOUR_CLIENT_ID',
        client_secret='YOUR_CLIENT_SECRET',
        region='mypurecloud.com'
    )

    # Example 1: Get conversations
    print('--- Getting Conversations ---')
    conversations = client.get_conversations()

    # Example 2: Create user
    print('\n--- Creating User ---')
    user = client.create_user('newuser@company.com', 'New User')

    # Example 3: Long-running job with periodic token refresh
    print('\n--- Simulating Long-Running Job ---')
    for i in range(5):
        print(f'Iteration {i+1}')
        conversations = client.get_conversations()
        # Token automatically refreshed if needed
```

Complete Node.js Example

```
const axios = require('axios');

class GenesysCloudAPI {
  constructor(clientId, clientSecret, region = 'mypurecloud.com') {
    this.clientId = clientId;
    this.clientSecret = clientSecret;
    this.region = region;
    this.authUrl = `https://login.${region}/oauth/token`;
    this.apiUrl = `https://api.${region}/api/v2`;

    this.accessToken = null;
    this.tokenExpiresAt = null;
  }

  // Step 1: Authenticate
  async authenticate(scopes = 'conversations:readonly users:readonly') {
    try {
      const response = await axios.post(this.authUrl, {
        grant_type: 'client_credentials',
        client_id: this.clientId,
        client_secret: this.clientSecret,
        scope: scopes
      });

      // Step 2: Receive token
      this.accessToken = response.data.access_token;

      // Expire 5 minutes early (proactive refresh)
      const expiresIn = response.data.expires_in;
      this.tokenExpiresAt = Date.now() + (expiresIn - 300) * 1000;

      console.log(`[${new Date().toLocaleTimeString()}] Authenticated successfully`);
      console.log(`Token expires at: ${new Date(this.tokenExpiresAt).toLocaleTimeString()}`);
      return true;

    } catch (error) {
```

```
    console.error('Authentication failed:', error.message);
    return false;
  }
}

// Ensure token is valid (refresh if needed)
async ensureToken(scopes = 'conversations:readonly users:readonly') {
  if (!this.accessToken) {
    return await this.authenticate(scopes);
  }

  // Proactive refresh if expiring
  if (Date.now() >= this.tokenExpiresAt) {
    console.log('Token expired, refreshing...');
    return await this.authenticate(scopes);
  }

  return true;
}

// Step 3: Use token to call API
async getConversations() {
  if (!await this.ensureToken()) {
    console.error('Failed to obtain access token');
    return null;
  }

  try {
    const response = await axios.get(`${this.apiUrl}/conversations`, {
      headers: {
        'Authorization': `Bearer ${this.accessToken}`,
        'Content-Type': 'application/json'
      },
      params: {
        pageSize: 100,
        pageNumber: 1
      }
    });

    console.log(`Retrieved ${response.data.entities.length} conversations`);
```

```
    return response.data;

  } catch (error) {
    if (error.response?.status === 401) {
      console.log('Token expired, retrying...');
      if (await this.authenticate()) {
        return await this.getConversations(); // Retry
      }
    }
    console.error('API call failed:', error.message);
    return null;
  }
}

// Example: Create user
async createUser(email, name) {
  if (!await this.ensureToken('users:manage')) {
    return null;
  }

  try {
    const response = await axios.post(`${this.apiUrl}/users`, {
      email: email,
      name: name,
      state: 'active'
    }, {
      headers: {
        'Authorization': `Bearer ${this.accessToken}`,
        'Content-Type': 'application/json'
      }
    });

    console.log(`Created user: ${response.data.name} (${response.data.id})`);
    return response.data;

  } catch (error) {
    console.error('Failed to create user:', error.message);
    return null;
  }
}
```

```
}

// Usage Example:
(async () => {
  const client = new GenesysCloudAPI(
    process.env.GENESYS_CLIENT_ID,
    process.env.GENESYS_CLIENT_SECRET,
    'mypurecloud.com'
  );

  // Example 1: Get conversations
  console.log('--- Getting Conversations ---');
  await client.getConversations();

  // Example 2: Create user
  console.log('\n--- Creating User ---');
  await client.createUser('newuser@company.com', 'New User');

  // Example 3: Long-running job
  console.log('\n--- Simulating Long-Running Job ---');
  for (let i = 0; i < 5; i++) {
    console.log(`Iteration ${i + 1}`);
    await client.getConversations();
  }
})();
```

Security Best Practices

Client Credentials Security Checklist:

□ Store client_secret in secure vault
  ├─ HashiCorp Vault
  ├─ AWS Secrets Manager
  ├─ Azure Key Vault
  └─ Never in code/git

□ Rotate credentials monthly

- [ ] Plan rotation schedule
- [ ] Generate new secret in Admin UI
- [ ] Update application config
- [ ] Verify working before removing old

□ Monitor usage patterns
- [ ] Track authentication frequency
- [ ] Alert on unusual activity
- [ ] Audit all API calls
- [ ] Review access logs

□ Implement audit logging
- [ ] Log authentication attempts
- [ ] Log API calls (not tokens)
- [ ] Track failures/errors
- [ ] Retain logs for compliance

□ Use HTTPS only
- [ ] All requests use HTTPS
- [ ] Validate SSL certificates
- [ ] Prevent man-in-the-middle
- [ ] Never fall back to HTTP

□ Limit scope to minimum
- [ ] Request only needed scopes
- [ ] Principle of least privilege
- [ ] Review scopes regularly
- [ ] Remove unused scopes

□ Implement error handling
- [ ] Handle authentication errors
- [ ] Retry failed requests
- [ ] Don't expose credentials in errors
- [ ] Log securely without tokens

□ Restrict permissions
- [ ] Assign minimal roles
- [ ] Use divisions for scope
- [ ] Review permissions quarterly
- [ ] Remove unneeded permissions

□ Monitor expiration
  ├─ Track token expiration
  ├─ Refresh proactively
  ├─ Alert if refresh fails
  └─ Implement fallback behavior

□ Version control security
  ├─ Never commit secrets
  ├─ Use .gitignore
  ├─ Use environment variables
  └─ Review git history

Common Use Cases

1. Salesforce ↔ Genesys Sync

Schedule: Every 30 minutes

Process:
├─ Authenticate with Client Credentials
├─ Query Salesforce API (with SF credentials)
├─ Fetch modified contacts
├─ Transform to Genesys format
├─ POST to Genesys externalcontacts API
├─ Log results
└─ Token auto-refreshes if needed

Benefits:
├─ No user interaction
├─ Fully automated
├─ Scheduled sync
└─ Deterministic refresh

2. Nightly Report Generation

Schedule: 2:00 AM daily

Process:
├─ Authenticate with Client Credentials
├─ Query analytics APIs
├─ Aggregate daily metrics
├─ Generate PDF report
├─ Email report to stakeholders
└─ Clean up temp files

Benefits:
├─ Runs while users sleep
├─ No performance impact
├─ Automated reporting
└─ Email delivery

3. Real-Time Data Processing

Schedule: Continuous

Process:
├─ Authenticate once (at startup)
├─ Proactively refresh before expiry
├─ Connect to event streaming
├─ Process events real-time
├─ Call APIs based on events
└─ Token automatically refreshed

Benefits:
├─ Always-on application
├─ Smooth operation
├─ No user waiting
└─ Fully automated

4. Bulk Contact Import

Schedule: Ad-hoc

Process:
├─ Authenticate with Client Credentials
├─ Read contact CSV file
├─ Parse and validate data
├─ Batch create in Genesys
├─ Log import results
├─ Handle errors gracefully
└─ Email summary report

Benefits:
├─ One-time operation
├─ No UI needed
├─ Simple authentication
└─ Bulk operations efficient

Comparison with Authorization Code

Client Credentials vs Authorization Code:

Timeline:
├─ Client Credentials: Milliseconds
├─ Authorization Code: Minutes (user delay)
└─ Different use cases

User Interaction:
├─ Client Credentials: None
├─ Authorization Code: Required (login, consent)
└─ Different workflows

Tokens:
├─ Client Credentials: No refresh token
├─ Authorization Code: Includes refresh token
└─ Different lifecycle management

Secret Exposure:

├─ Client Credentials: Secret sent to server
├─ Authorization Code: Secret never exposed to browser
└─ Different security models

Use Case:
├─ Client Credentials: Background jobs, services
├─ Authorization Code: Web/mobile apps, users
└─ Choose based on scenario

When Choose Client Credentials:
├─ No user interaction needed
├─ Automated/scheduled processes
├─ Service-to-service integration
├─ No user-specific access needed
└─ Application acts as itself

When Choose Authorization Code:
├─ User logs in to app
├─ User grants permission
├─ Long-lived access needed
├─ User-specific data required
└─ User can revoke access anytime

Key Takeaways: Chapter 3

- **Single-Step Process** - Direct authentication, no user interaction
- **No User Context** - Application acts as itself (no /users/me access)
- **Service-to-Service** - Primary use for automated integrations
- **No Refresh Token** - Simply authenticate again when token expires
- **Immediate** - Token available instantly for use
- **Fully Automated** - Perfect for background jobs and scheduled tasks
- **Simple Flow** - One request for token, then use it
- **Role-Based Access** - Permissions determined by assigned roles/divisions

---

Interview Prep: Client Credentials Grant

Question	Answer
When use Client Credentials?	Service-to-service, background jobs, no user interaction
Single-step or two-step?	Single-step (direct authentication)
Refresh token included?	No - authenticate again when token expires
User context?	No (/v2/users/me not available)
Client_secret exposure?	Only server-to-server (HTTPS required)
How determine permissions?	OAuth client roles and scope settings
Typical use case?	Salesforce sync, nightly reports, data imports
Token lifetime?	Default 1 hour (configurable 300-172,800 seconds)
HTTPS required?	Yes (always)
Error 401 handling?	Authenticate again with fresh credentials

---

Document Version

Chapter: 3 of 8
Last Updated: March 2026
Status: Current with RFC 6749
Scope: Client Credentials Grant, Service Integration, Implementation

---

Revision #1
Created 14 March 2026 19:31:50 by Cesar Gzz
Updated 14 March 2026 19:32:03 by Cesar Gzz