Prise en Main de l'API

Table des matières

Introduction 1
Demande d'identifiants 2
Authentification 3
Liste des endpoints 5
Actualisation des données d'authentification (/refresh) 6
Récupération des données d'une table 7
Erreurs de retour 9
Procédure de signature 10
Conditions d'utilisation 15

Introduction

L'API LDC a été conçue dans le but de simplifier l'accès aux données stockées dans les différentes bases Génésis. Elle permet d'interroger facilement les tables à l'aide de paramètres définis, pour filtrer les résultats selon des critères précis.

Ce guide vous accompagnera à travers les étapes suivantes :

- Mise à jour de vos informations d'authentification
- Récupération des données depuis une table cible
- Développement Java d'un appel à l'API

Vous trouverez à la fin de ce document nos conditions d'utilisation.

Demande d'identifiants

Toute demande d'identifiants d'accès doit être effectuée via le catalogue de services (lien).

Le client souhaitant consommer l'API doit également signer une décharge de responsabilité en cas d'indisponibilité du service. Cette décharge est transmise par le responsable du bureau d'études, Julien GRELLIER.

Authentification

L'accès à l'API LDC nécessite une authentification basée sur deux éléments :

- **Clé API** : fournie par LDC, elle est valide pendant un an.
- **Signature** : un hachage de la requête

Un mail vous sera envoyé une semaine avant l'expiration de votre clé API, pour vous rappeler d'actualiser vos données d'authentification. De plus, un warning sera présent dans les réponses retournées lorsque la date d'expiration est inférieure à 7 jours.

En-têtes HTTP requis

Lors de chaque appel à l'API, les éléments suivants doivent être renseignés dans les en-têtes HTTP :

- **X-API-Key** : votre clé API
- **X-API-Signature** : la signature calculée à partir de la clé et du secret
- **X-Timestamp** : date au format ISO 8601 (RFC3339) correspondant à la date au moment de l'appel
- **X-Nonce** : identifiant unique de requête (UUID v4), utilisé pour prévenir les attaques par rejeu

Gestion des erreurs d'authentification

En cas d'erreur d'authentification, l'API retourne une réponse explicite :

Clé API invalide (expirée ou incorrecte)

```
{
    "success": false,
    "error": {
        "code": "ERR_INVALID_API_KEY",
        "message": "La clé API n'est pas valide."
    }
}
```

Signature incorrecte :

```
{
    "success": false,
    "error": {
        "code": "ERR_INVALID_API_SIGNATURE",
        "message": "La signature n'est pas valide."
    }
}
```

Exemple de warning d'expiration :


```
{
    "success": true,
    "data": {
        "SOCIETE": "SNV",
        "NUM_SOCIETE": "5"
    },
    "warning": "La date d'expiration de la clé est inférieure à une semaine. Da
te d'expiration : 2025-11-15 11:44:00"
}
```

Liste des endpoints

Route	Description	Auth	Paramètres
/refresh	Rafraîchit les jetons liés à une clé API (nouvelle clé API + nouveau secret) pour une durée de 12 mois.	Obligatoire	Aucun
/get /post	Récupère les données d'une table en fonction des paramètres fournis.	Obligatoire	Lien vers le paramétrage

Actualisation des données d'authentification (/refresh)

Un endpoint GET /refresh est disponible. Lorsqu'il est appelé de manière authentifiée, un mail contenant votre nouvelle clé API vous sera envoyé. Le nouveau secret associé vous sera retourné dans la réponse à cet appel. C'est le seul moment où cette information vous sera communiquée.

URL de la requête : [Url de base de l'application Génésis]-api/refresh

Exemple : https://arc05402-api.svlidc.com/gaston/api/refresh

Réponse JSON

```
{
    "API-SECRET" : "Nouveau secret"
}
```

Mail envoyé



Récupération des données d'une table

La récupération des données se fait avec les paramètres suivants :

Paramètres pour un appel GET	Paramètres pour un appel POST
table : nom de la table à interroger	**table** : nom de la table à interroger
soc : code société	**soc** : code société
lang : langue dans laquelle on souhaite recevoir les enregistrements (fonctionne uniquement si la table est internationalisée)	**langue** : langue dans laquelle on souhaite recevoir les enregistrements (fonctionne uniquement si la table est internationalisée)
s_cols : colonnes à sélectionner	**selCols** : colonnes à sélectionner
c_cols : colonnes à utiliser dans les critères	**critCols** : colonnes à utiliser dans les critères
c_opes : opérateurs des critères	**critOpes** : opérateurs des critères
c_vals : valeurs associées aux critères	**critVals** : valeurs associées aux critères
t_cols : colonnes utilisées pour le tri	**triCols** : colonnes utilisées pour le tri
t_opes : opérateurs de tri	**triOpes** : opérateurs de tri
page : numéro de page	**page** : numéro de page
page_size : taille de la page	**pageSize** : taille de la page

Liste des opérateurs critères valides :

=	!=	>	>=	<	<=	BETWEEN	NOT BETWEEN	LIKE	NOT LIKE	IN	NOT IN

Liste des opérateurs de tri valides :

asc	desc

Le paramètre **table** est le seul requis. Les autres sont facultatifs. Lors d'un appel GET, les valeurs multiples sont séparées par un pipe « **|** ». Chaque colonne définie dans **c_cols** est associée à l'opérateur correspondant de **c_opes** et à la valeur correspondante dans **c_vals** (ex. : la première colonne utilise le premier opérateur et la première valeur, etc.).

L'ordre des paramètres dans l'URL n'influence pas le traitement. Il est également possible d'effectuer une recherche multiple sur un même champ numérique en séparant les valeurs par des virgules.

Le format de date doit être au format ISO 8601 avec fuseau horaire obligatoire (RFC 3339), par exemple : `2026-03-26T14:24:28Z` ou `2026-03-26T16:28:12+02:00`.

Exemple d'utilisation

Exemple d'url (/get)

*?**table**=T_SOCIETE&**s_cols**=NUM_SOCIETE|SOCIETE&**c_cols**=NUM_SOCIETE|SOCIETE&**c_opes**==|=&**c_vals** =5|SNV&**t_cols**=NUM_SOCIETE&**t_opes**=asc&**page**=1&**page_size**=1*

Exemple de body (/post)

```
{
    table=T_SOCIETE,
    selCols=[NUM_SOCIETE, SOCIETE],
    critCols=[NUM_SOCIETE, SOCIETE],
    critOpes=[=, =],
    critVals=[5, SNV],
    triCols=NUM_SOCIETE,
    triOpes=ASC,
    page=1,
    pageSize=1,
    soc=1,
    langue=”FR”
}
```

Traduction SQL

***SELECT** NUM_SOCIETE, SOCIETE **FROM** T_SOCIETES **WHERE** NUM_SOCIETE=5 AND SOCIETE='SNV' AND SOC=1 AND LANGUE=”FR” **ORDER BY** NUM_SOCIETE ASC* **LIMIT** 1 **OFFSET** 1

Réponse JSON


```
{
    "success": true,
    "data": {
        "SOCIETE": "SNV",
        "NUM_SOCIETE": "5"
    },
    "pagination": {
        "page": 1,
        "pageSize": 1,
        "totalItems": 1,
        "totalPages": 1,
        "links": {
            "first": "http://localhost:30102/gaston/api/get?table=T_SOCIETE&s_cols=NUM_SOCIETE|SOCIETE&c_cols=NUM_SOCIETE|SOCIETE&c_opes==|=&c_vals=5|SNV&t_cols=NUM_SOCIETE&t_opes=asc&page=1&page_size=1",
            "last": "http://localhost:30102/gaston/api/get?table=T_SOCIETE&s_cols=NUM_SOCIETE|SOCIETE&c_cols=NUM_SOCIETE|SOCIETE&c_opes==|=&c_vals=5|SNV&t_cols=NUM_SOCIETE&t_opes=asc&page=1&page_size=1"
        }
    }
}
```

Erreurs de retour

La gestion d'erreur de l'API utilise les codes d'erreur HTTP standards, voici les différentes erreurs publiques :

404 - BAD_REQUEST : Requête invalide	
ERR_DEFAULT	Vous n'êtes pas autorisé à faire cette action.
ERR_INVALID_API_KEY	La clé API n'est pas valide.
ERR_INVALID_API_SIGNATURE	La signature n'est pas valide.
ERR_LIST_SIZE_MISMATCH	Le nombre de colonnes, d'opérateurs et de valeurs ne correspond pas.
ERR_METAEXCEPTION	Une MetaException a été levée. Une erreur s'est produite.
ERR_SORT_LIST_SIZE_MISMATCH	La liste de colonnes de tri n'est pas de la même taille que la liste d'opérateurs de tri.
ERR_TABLE_NAME_MISSING	Le nom de la table n'est pas fourni.
ERR_UNAUTHORIZED_OPERATOR	La liste d'opérateurs contient des opérateurs non autorisés.
ERR_BAD_DATE_FORMAT	La date doit être fournie au format ISO 8601 (RFC3339) (ex : 2025-10-21T14:30:00Z)
ERR_PAGE_TYPE_MISMATCH	Le type pour le paramètre 'page' doit être un entier supérieur à 0.
ERR_PAGE_SIZE_TYPE_MISMATCH	Le type pour le paramètre 'page_size' doit être un entier supérieur à 0.
ERR_TOO_MANY_RECORDS	Le nombre d'enregistrements retournés doivent être <= 1000
ERR_INVALID_JSON	Le corps de la requête est mal formé et ne respecte pas le format JSON attendu. Vérifiez la syntaxe du JSON.
401 - UNAUTHORIZED : Permissions insuffisantes	
ERR_INVALID_API_SIGNATURE	La signature n'est pas valide.
ERR_INVALID_API_KEY	La clé API n'est pas valide.
429 - TOO_MANY_REQUESTS : Nombre de requêtes dépassés	
ERR_TOO_MANY_REQUESTS	Le nombre maximal de requêtes autorisées a été atteint. Veuillez réessayer ultérieurement.
503 - SERVICE_UNAVAILABLE : L'API ne répond pas	

Procédure de signature

La procédure ci-dessous permet de générer la signature :

1. Principe général

Chaque requête doit être signée avec un HMAC-SHA256 basé sur :

- La méthode http
- Le schéma
- L'host
- Le chemin
- Les paramètres triés
- Le timestamp ISO 8601 (RFC3339)
- Le nonce (UUID v4)
- Le hash du body (POST uniquement)

La signature finale est envoyée dans le header X-API-Signature.

2. Étapes de génération

Étape 1 — Générer le timestamp

Format ISO 8601 UTC, ex. : 2025-11-12T11:30:00Z → Header : X-Timestamp

Format accepté :

Entrée client	Interprétation backend
`2026-03-26T14:24:28Z`	UTC direct
`2026-03-26T14:24:28+00:00`	UTC équivalent
`2026-03-26T16:28:12+02:00`	Converti en UTC
`2026-03-26T16:28:12`	Rejeté (pas de timezone)

Étape 2 — Construire la canonical query string

1. Récupérer tous les paramètres de l'URL
2. Encoder nom + valeur en **RFC 3986**
3. Pour chaque valeur → une paire nom=valeur
4. Trier toutes les paires par ordre ASCII
5. Concaténer avec &

Exemple :

```
s_cols=SOCIETE&table=T_SOCIETE
```

Étape 3 — Calculer le hash du body (POST)

1. Prendre le corps en UTF-8
2. Retirer le BOM UTF-8 s'il existe
3. Retirer le CRLF final s'il existe
4. SHA-256(body) → Base64

Exemple :

```
contentSha256 = "u5H8mWq3…"
```

Pour GET : valeur vide.

Étape 4 — Construire la string à signer

Format exact :

```
<VERB>
<schéma>://<host>
<path>
<canonicalQueryString>
<timestamp>
<nonce>
<contentSha256>
```

Exemple :

```
GET
https://arc50482-api.svildc.com
/get
s_cols=SOCIETE&table=T_SOCIETE
2025-11-12T11:30:00Z
1f1fb752-e1e7-4690-a0a3-6821d121261a
```

Étape 5 — Calculer la signature HMAC-SHA256

signature = Base64(HMAC_SHA256(secret, stringToSign)) → Header : X-API-Signature: <signature>

3. Pseudo-code résumé

```
timestamp = nowUtcIso8601()
Nonce = generateNonce()
canonicalQS = canonicalizeQuery(params)
contentHash = (POST ? sha256Base64(normalize(body)) : "")

stringToSign =
  method + "\n" +
  scheme + "://" + lower(host) + "\n" +
  path + "\n" +
  canonicalQS + "\n" +
  timestamp + "\n" +
  nonce + "\n" +
  contentHash

signature = hmacSha256Base64(secret, stringToSign)

Headers:
  X-API-KEY       = apiKey
  X-Timestamp     = timestamp
  X-API-Signature = signature
  X-Nonce = nonce
```

4. Exemple minimal

```
GET /get?s_cols=SOCIETE&table=T_SOCIETE

StringToSign =
GET
https://arc50482-api.lesidc.com
/get
s_cols=SOCIETE&table=T_SOCIETE
2025-11-12T11:30:00Z
1f1fb752-e1e7-4690-a0a3-6821d121261a



Signature = Base64(HMAC_SHA256(secret, StringToSign))
Headers envoyés :
X-API-KEY: ...
X-Timestamp: 2025-11-12T11:30:00Z
X-API-Signature: <signature>
X-Nonce: <nonce>
```

Nous allons maintenant explorer différentes implémentations permettant d'appeler l'API.

Ici, les méthodes *getApiKey()* et *getApiSecret()* ont pour rôle de récupérer respectivement la clé et le secret API. Ces méthodes doivent être implémentées de manière à accéder aux emplacements où ces informations sont stockées.

Appel GET – Spring (org.springframework.web.client.RestTemplate)

```
public static String getSocietes() {
    // Pour un appel à  https://lab501-arc05402-api.lesidc.com/get?table=T_SOCIETE...&t_opes=asc
    // ============================
    // 1) Définition des éléments fixes
    // ============================
    String scheme = "https";
    String host   = "lab501-arc05402-api.lesidc.com";
    String path   = "/get";
    String method = "GET";

    // ============================
    // 2) Construction des paramètres de la query string
    // ============================
    Map<String, List<String>> params = new HashMap<>();
    params.put("table", List.of("T_SOCIETE"));
    params.put("s_cols", List.of("NUM_SOCIETE"));
    params.put("c_cols", List.of("NUM_SOCIETE|SOCIETE"));
    params.put("c_opes", List.of("=|="));
    params.put("c_vals", List.of("5|SNV"));
    params.put("t_cols", List.of("NUM_SOCIETE"));
    params.put("t_opes", List.of("asc"));

    // 2.a Canonical query string (RFC 3986 + tri)
    String canonicalQS = canonicalize(params);

    // 2.b URL finale à appeler
    String url = scheme + "://" + host + path + "?" + canonicalQS;

    // ============================
    // 3) Génération du timestamp (horodatage anti-rejeu) et du nonce
    // ============================
    String timestamp = java.time.OffsetDateTime
            .now(java.time.ZoneOffset.UTC)
            .toString(); // ex : 2025-11-12T11:30:00Z
     String nonce = generateUuidV4();

    // ============================
    // 4) Construction de la stringToSign
    // ============================
    String contentSha256 = ""; // GET : pas de body => chaîne vide

    String stringToSign =
            method + "\n" +
            scheme + "://" + host.toLowerCase() + "\n" +
            path + "\n" +
            canonicalQS + "\n" +
            timestamp + "\n" +
```

```
            nonce + "\n" +
            contentSha256;


    // ============================
    // 5) Récupération des secrets + calcul de la signature
    // ============================
    String apiKey      = getApiKey();
    String apiSecret   = getApiSecret();
    String apiSignature;

    try {
        apiSignature = hmacSha256Base64(stringToSign, apiSecret);
    } catch (Exception e) {
        LOGGER.error("Erreur lors du calcul de la signature", e);
        return "Erreur locale de signature";
    }

    // ============================
    // 6) Construction des headers HTTP
    // ============================
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.set("X-API-Key", apiKey);
    headers.set("X-API-Signature", apiSignature);
    headers.set("X-Timestamp", timestamp);
    headers.set("X-Nonce", nonce);

    HttpEntity<Void> entity = new HttpEntity<>(headers);

    // ============================
    // 7) Appel à l'API
    // ============================
    try {
        ResponseEntity<String> response =
                restTemplate.exchange(url, HttpMethod.GET, entity, String.class);

        return response.getBody();
    } catch (HttpClientErrorException | HttpServerErrorException e) {
        LOGGER.error("Erreur HTTP 4XX / 5XX", e);
        return e.getResponseBodyAsString();
    } catch (ResourceAccessException e) {
        LOGGER.error("API inaccessible", e);
        return "API inaccessible";
    }
}
```

Conditions d'utilisation

L'accès à l'API LDC est soumis à des quotas définis au cas par cas. Ces quotas peuvent s'appliquer à différentes échelles de temps : par seconde, minute, heure, jour, mois ou année.

En cas de dépassement des limites autorisées, l'API renverra une erreur **HTTP 429 – TOO_MANY_REQUESTS**. Dans ce cas, toute tentative ultérieure sera bloquée jusqu'à ce qu'un nouveau seuil d'autorisation soit atteint.

Si l'API est injoignable car le serveur qui l'héberge est en maintenance, **tous les appels retourneront une erreur HTTP 503 – SERVICE_UNAVAILABLE**. Ces requêtes ne seront pas traitées ultérieurement après le redémarrage du serveur.

Il est impératif que l'algorithme qui doit être utilisé pour la création de la signature soit l'algorithme SHA256. En effet, il est utilisé côté serveur pour vérifier l'authenticité de la clé transmise. Un algorithme différent entraînera un échec de validation.