Guide utilisateur

Web service REST presenceRegistration

Check In and Out at Work

Version	Date	Nature des modification
1.0	01/03/2024	Première version
1.1	03/06/2024	Réécriture des explications concernant OAuth

Contenu

Enregistrement des présences dans le secteur du nettoyage

Conformément à la loi programme du 26/12/2022, les travailleurs effectuant des activités spécifiques de nettoyage doivent obligatoirement enregistrer le début (IN) et la fin (OUT) de leurs prestations, ainsi que les périodes de repos.

Un enregistrement de présence représente l'entrée ou la sortie d'un employé sur son lieu de travail. Il existe donc 2 types de présence :

- **IN** : l'entrée du travailleur (mais également la fin d'une pause)
- **OUT** : la sortie du travailleur (mais également le début d'une pause)

Un enregistrement de présence typique regroupe les informations suivantes :

- **QUI** ? Qui est entré ou sorti du bâtiment ? Cette information est représentée par le numéro d'identification à la sécurité sociale (NISS) du travailleur.
- **OÙ** ? Sur quel lieu de travail le travailleur est-il entré ou d'où est-il sorti ? Cette information est représentée soit par les coordonnées de géolocalisation du travailleur au moment du 'check in' (entrée) ou du 'check out' (sortie), ou d'une description de l'adresse au format texte.
- **QUAND** ? À quelle heure le travailleur est-il entré ou sorti ? Cette information est représentée par une date et une heure, auxquelles le travailleur a effectué son check in ou son check out.

Check **I**n **a**nd **O**ut at Work (parfois abrégé 'CIaO') est le service en ligne de suivi des prestations des travailleurs des différents secteurs concernés. Elle permet la création et la consultation de 'presence registrations' (enregistrements de présences).

Deux méthodes permettent l'enregistrement des présences :

- enregistrement à l'aide d'un smartphone, et
- appel à un web service REST (couplé à votre éventuel système de badge existant), appelé aussi **PresenceRegistrations**.

Ce guide utilisateur décrit les aspects fonctionnels du web service REST.

Service SOAP et service REST

Le portail de la sécurité sociale propose deux services avec le nom **PresenceRegistrations** :

- l'un utilise la technologie SOAP, et
- l'autre utilise la technologie REST.

Il s'agit de deux web services différents, destinés à des secteurs particuliers :

- Les secteurs de la **construction**, de la **viande** et du **gardiennage** doivent utiliser le service **SOAP**. L'enregistrement de présence se fait une fois par jour, par personne et par déclaration de travail.
- Le secteur du **nettoyage** doit utiliser le service **REST**. L'enregistrement de présence se fait à chaque entrée et sortie du travailleur, et également à chaque pause pour chaque lieu de travail.

Au sein d'un même endroit, il peut y avoir des travailleurs qui doivent utiliser le service REST plusieurs fois par jour et d'autres qui doivent utiliser le service SOAP une fois par jour. Par exemple, dans le cadre d'un chantier de construction qui inclus des prestations de nettoyage.

D'autres secteurs seront ajoutés au cours de l'année 2024 et 2025. Ils devront également utiliser le service REST.

Comme mentionné ci-dessus, ce guide utilisateur décrit uniquement le service REST.

Pour le service SOAP, rendez-vous sur la page 'Comment enregistrer les présences ?' de Checkinatwork sur le portail de la sécurité sociale.

Préparation

Avant d'accéder au service REST **PresenceRegistrations**, vous devez effectuer certaines démarches préalables. Si vous avez déjà utilisé un web service de la sécurité sociale, il est possible que certaines de ces étapes aient déjà été effectuées.

S'identifier comme employeur

Pour utiliser les services en ligne de la sécurité sociale, l'entreprise doit être reconnue en tant qu'employeur sur le portail de la sécurité sociale. Le portail regroupe l'ensemble des services en ligne de la sécurité sociale.

Afin de protéger les données des employeurs et des travailleurs, vous devez, en tant qu'utilisateur des services en ligne :

- être identifié en tant qu'employeur, et
- être enregistré sur le portail.

Pour ce faire, suivez les quatre étapes décrites sur la page Identifiez-vous en tant qu'employeur sur le portail de la sécurité sociale.

Demander ou créer un certificat

Pour invoquer un service REST de la sécurité sociale, vous devez utiliser un certificat.

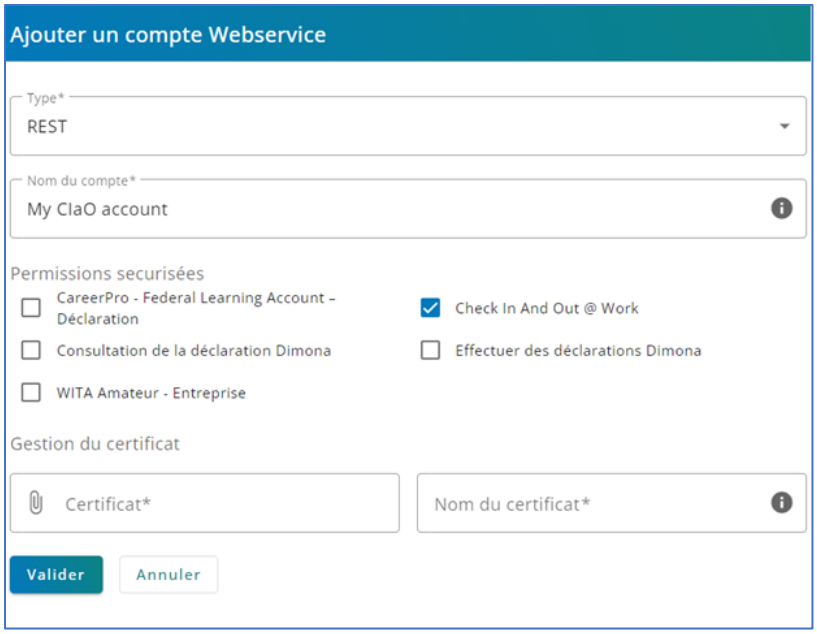
Il n'est plus obligatoire d'utiliser un certificat GlobalSign pour accéder aux services REST de la sécurité sociale. Vous pouvez utiliser un certificat auto-signé (self-signed).

Créer un compte web service dans Chaman

Chaman ou Channel Management est le service en ligne qui permet de gérer les canaux techniques (FTP, SFTP, SOAP, et REST) de la sécurité sociale.

Un manuel est disponible sur le portail de la sécurité sociale.

Dans le service en ligne Chaman, vous devez créer un compte web service de type REST, sélectionner la permission 'Check in And Out at Work' et y enregistrer votre certificat.



- Utilisez le lien suivants [Chaman : gestion des canaux techniques]

Demander un jeton d'accès OAuth

Les API REST du portail de la Sécurité Sociale utilisent le protocole de sécurité OAuth2. Votre application doit obtenir un jeton de la part du serveur d'autorisation OAuth2 de la Sécurité Sociale.

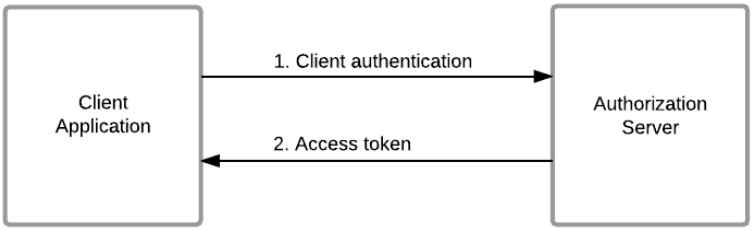
Vous devez pour cela utiliser le protocole d'identification clients OAuth2 (plus d'information sur [cette page du site IETF]).

Pour vous aider dans votre intégration, nous intégrons ci-dessous les informations pertinentes.

Requête du jeton d'accès

L'authentification client inclus les étapes suivantes :

1. Le client s'authentifie auprès du serveur d'autorisation et demande un jeton d'accès au point de terminaison du jeton.
2. Le serveur d'autorisation authentifie le client et, s'il est valide, émet un jeton d'accès.



Le client doit faire une requête vers le endpoint en ajoutant les paramètres suivant, avec le format « **application/x-www-form-urlencoded** », et l'encodage **UTF-8** dans le http request entity-body :

Nom du paramètre	Obligatoire	Valeur attendue
client_assertion	Oui	Un JWT signé qui authentifie le client avec le serveur d'autorisation comme décrit dans RFC 7519 du site IETF.org Le contenu de ce JWT est décrit dans la section 'Authentification du client' ci-dessous.
client_assertion_type	Oui	La valeur doit être : « **urn:ietf:params:oauth:client-assertion-type:jwt-bearer** ».
grant_type	Oui	La valeur doit être : « **client_credentials** ».
scope	Non	Il s'agit du scope de la requête d'accès. Si non remplie, le scope par défaut de votre client sera sélectionné.

La requête pour l'obtention du jeton d'accès doit être soumise à ce endpoint :

https://services.socialsecurity.be/REST/oauth/v5/token

Authentification client

Il s'agit d'un JWT signé qui authentifie le client avec le serveur d'autorisation. Les valeurs attendue dans le payload (contenu) de ce JWT sont décrites dans le tableau suivant.

Nom de l'attribut	Obligatoire	Type	Valeur attendue
Jti	Oui	String	'jti' (JWT ID), un identifiant unique pour le JWT
iss	Oui	String ou URI	Signifie 'issuer'. La valeur doit être le 'Client ID' généré dans Chaman (*self_service_chaman_xxxxxx*).

sub	Oui	String ou URI	Signifie 'subject'. La valeur doit être le 'Client ID' généré dans Chaman (*self_service_chaman_xxxxxx*).
aud	Oui	String ou URI	Signifie 'audience'. La valeur doit être : **https://services.socialsecurity.be/REST/oauth/v5/token**
exp	Oui	Nombre (date numérique en secondes)	Signifie 'expiration time', c'est-à-dire le moment à partir duquel le JWT ne doit plus être accepté. Le site https://unixtime.org peut vous aider à visualiser le format attendu
nbf	Non	Nombre (date numérique en secondes)	Signifie 'not before'. c'est-à-dire le moment à partir duquel le JWT doit être accepté.
iat	Non	Nombre (date numérique en secondes)	Signifie 'issued at'. C'est-à-dire le moment où le JWT est fait.

La signature doit être créée en utilisant le certificat (et la clé privée correspondant) qui a été envoyée sur Chaman.

Réponse

Si l'authentification est un succès, vous recevrez un jeton d'accès. Le serveur OAuth de la sécurité sociale retourne un jeton 'Bearer' comme décrit dans RFC 6750.

Attention : Les jeton d'accès de la Sécurité Sociale ont une période de validité de **10 minutes**.

Votre application client doit gérer cela et demander un autre jeton lorsque le jeton actuel est sur le point d'expirer.

Eviter cependant de renouveler le jeton si cela n'est pas nécessaire. Ne renouvelez celui-ci que s'il expire dans moins d'une minute.

Exemple de code Java

Voici un exemple de code en Java commenté permettant d'obtenir un jeton d'accès de la part du serveur OAuth de la Sécurité Sociale.

```
package be.smals.art30bister;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.Base64;
import java.util.Properties;
import java.util.UUID;
import java.util.logging.Level;
import java.util.logging.Logger;

public class GetAccessToken {

    private static final Logger logger =
Logger.getLogger(GetAccessToken.class.getName());

    /**
     * Main method to generate an OAuth 2.0 token using client credentials grant.
     * To run this program, provide the path to a properties file as a command line
argument.
     * The properties file should contain the following keys (and be adjusted
accordingly to your environment)
     * audience=https://services.socialsecurity.be/REST/oauth/v5/token
     * authUrl=https://services.socialsecurity.be/REST/oauth/v5/token
     * clientId=self_service_chaman_xxxxxxxxxxxxxxx  // replace with your client ID
     * scope=scope:rsz-onss:gestion:check-in-and-out-work-rest:enterprise // replace
with the scope provided in the documentation
     * password=your-key-store-password // replace with your keystore password
     * certificate=path-to-your-PKCS12-file // replace with the path to your PKCS12 file
     * keyAlias=your key alias // replace with your key alias
     *
     * @param args Command line arguments should include the path to the properties
file.
     */
    public static void main(String[] args) {
        Properties prop = new Properties();
        try (FileInputStream fis = new FileInputStream(args[0])) {
            prop.load(fis);
        } catch (IOException e) {
            logger.log(Level.SEVERE, e.getMessage(), e);
            return;
        }
```

```
        // Extract necessary properties for OAuth token request
        String audience = prop.getProperty("audience");
        String authUrl = prop.getProperty("authUrl");
        String clientId = prop.getProperty("clientId");
        String scope = prop.getProperty("scope");
        String password = prop.getProperty("password");
        String certificate = prop.getProperty("certificate");
        String keyAlias = prop.getProperty("keyAlias");

        try {
            // Generate and print the access token
            String accessTokenResponse = getAccessToken(certificate, password, keyAlias,
audience, clientId, authUrl, scope);
            logger.log(Level.INFO, accessTokenResponse);
        } catch (Exception ex) {
            logger.log(Level.SEVERE,ex.getMessage(), ex);
        }
    }

    /**
     * Retrieves the access token using the provided parameters.
     *
     * @param certificate Path to the certificate file.
     * @param password Certificate's password.
     * @param keyAlias Alias of the private key within the certificate.
     * @param audience Token endpoint audience.
     * @param clientId Client ID for OAuth.
     * @param authUrl Authorization server URL.
     * @param scope OAuth scopes requested.
     * @return A string containing the access token.
     * @throws IOException If an I/O error occurs during communication with the OAuth
server.
     * @throws GeneralSecurityException If there is an issue with the private key or
signature.
     * @throws URISyntaxException If the URI is invalid.
     */
    private static String getAccessToken(String certificate, String password, String
keyAlias, String audience, String clientId, String authUrl, String scope)
            throws IOException, GeneralSecurityException, URISyntaxException {
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        PrivateKey privateKey;

        // Load the keystore and extract the private key
        try (FileInputStream fis = new FileInputStream(certificate)) {
            keyStore.load(fis, password.toCharArray());
            privateKey = (PrivateKey) keyStore.getKey(keyAlias, password.toCharArray());
        }

        // Create a signed JWT for the OAuth authentication request
        String jwt = createJWT(privateKey, clientId, audience);

        // Send the JWT to the OAuth server and retrieve the access token
        return doPostToAuthServer(jwt, new URI(authUrl), scope);
    }
```

```
    /**
     * Creates a signed JWT for the OAuth authentication request.
     *
     * @param privateKey The private key used to sign the JWT.
     * @param clientId The OAuth client ID.
     * @param audience The audience for the token, usually the token URL.
     * @return A signed JWT string.
     * @throws NoSuchAlgorithmException If the RSA algorithm is not supported.
     * @throws InvalidKeyException If the private key is invalid.
     * @throws SignatureException If there is an issue with the signature.
     * @throws InvalidKeyException If the private key is invalid.
     */
    private static String createJWT(PrivateKey privateKey, String clientId, String
audience) throws NoSuchAlgorithmException, InvalidKeyException, SignatureException {
        long nowMillis = System.currentTimeMillis(); // Current time in milliseconds
        long expMillis = nowMillis + 3600000; // JWT validity for 1 hour
        String jti = UUID.randomUUID().toString(); // Unique identifier for each JWT

        // Create the JWT header and payload
        String header = "{\"alg\":\"RS256\",\"typ\":\"JWT\"}"; // JWT header
        String payload = String.format(

"{\"iss\":\"%s\",\"sub\":\"%s\",\"aud\":\"%s\",\"exp\":%d,\"iat\":%d,\"jti\":\"%s\"}",
                clientId, clientId, audience, expMillis / 1000, nowMillis / 1000, jti);

        // Encode the header and payload
        String encodedHeader =
Base64.getUrlEncoder().withoutPadding().encodeToString(header.getBytes(StandardCharsets.
UTF_8));
        String encodedPayload =
Base64.getUrlEncoder().withoutPadding().encodeToString(payload.getBytes(StandardCharsets
.UTF_8));

        // Concatenate the encoded header and payload
        String assertion = encodedHeader + "." + encodedPayload;

        // Sign the JWT using the RS256 algorithm
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
        signature.update(assertion.getBytes(StandardCharsets.UTF_8));
        byte[] signedAssertion = signature.sign();

        // Encode the signature and append it to the assertion to create the final JWT
        String encodedSignature =
Base64.getUrlEncoder().withoutPadding().encodeToString(signedAssertion);
        return assertion + "." + encodedSignature;
    }

    /**
     * Sends a POST request to the OAuth server and retrieves the access token.
     *
     * @param jwt The JWT used for client assertion.
     * @param authUri The URI of the OAuth server.
     * @param scope The requested scope(s).
     * @return The access token as a String.
```

```
     * @throws IOException If an I/O error occurs during communication with the OAuth
server.
     */
    private static String doPostToAuthServer(String jwt, URI authUri, String scope)
throws IOException {
        // Create a connection to the OAuth server
        URL url = new URL(authUri.toString());
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded");
        connection.setDoOutput(true);

        // Construct the request parameters
        String params =
String.format("grant_type=%s&client_assertion_type=%s&client_assertion=%s&scope=%s",
                "client_credentials", "urn:ietf:params:oauth:client-assertion-type:jwt-
bearer", jwt, scope);

        // Send the request and read the response
        try (DataOutputStream wr = new DataOutputStream(connection.getOutputStream())) {
            wr.writeBytes(params);
            wr.flush();
        }

        // Read the response from the server
        StringBuilder response = new StringBuilder();
        if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
                reader.lines().forEach(response::append);
            }
        } else {
            // Read the error stream to capture any error messages from the server
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getErrorStream()))) {
                reader.lines().forEach(response::append);
            }
        }

        return response.toString();
    }


}
```

Ce code est fourni as-is. Nous ne donnons pas d'exemple dans d'autres langages de programmation et nous ne fournissons pas d'assistance dans l'élaboration de votre code.

Appel de l'API REST PresenceRegistration

Après avoir obtenu votre jeton d'accès de la part du serveur d'autorisation de la Sécurité Sociale, vous pouvez désormais appeler le service REST PresenceRegistration en transmettant le jeton reçu.

Votre application client doit utiliser la méthode 'Authorization Request Header Field' décrit dans la section 2.1 du RFC 6750.

Swagger

Le Swagger est un fichier au format YAML, un standard de représentation de données. Le Swagger de PresenceRegistration décrit les méthodes et les zones. Vous le trouverez sur le portail de la sécurité sociale, en bas de la page suivante :

https://www.socialsecurity.be/site_fr/employer/applics/check-in-and-out-at-work/general/how-to-register.htm

Principe de fonctionnement

L'ONSS met à disposition des employeurs un web service, leur permettant :

- d'enregistrer les présences de leurs travailleurs dans ClaO.
- de consulter les présences de leur travailleurs, et de consulter les éventuelles remarques.

Environnements

L'ONSS met à disposition 2 environnements dont voici les endpoints :

Environment	Adresse
Simulation	https://services-sim.socialsecurity.be/REST/presenceRegistration/v1
Production	https://services.socialsecurity.be/REST/presenceRegistration/v1

Vos tests doivent être réalisés sur l'environnement de Simulation.

L'environnement de Production ne doit contenir que des données de prestations réellement effectuées.

Editeurs de software

Cette partie s'adresse aux éditeurs de software qui souhaitent proposer leurs services aux entreprises concernées pas le Check In & Out at Work.

Votre service consiste-t-il uniquement à envoyer des enregistrements vers la Sécurité Sociale (par exemple, vous gérez une badgeuse) ?

Dans ce cas, vous pouvez utiliser votre propre certificat.

Souhaitez-vous offrir des fonctionnalités avancées impliquant la consultations des enregistrements et de leurs anomalies ?

Dans ce cas, vous devrez utiliser le certificat de votre client.

Vous devrez assister votre client pour la création d'un certificat self-signed, et l'envoi de celui-ci sur Chaman.

Ensuite, vous pourrez installer votre logiciel sur une machine appartenant au client.

Asynchronisme

Il convient de souligner que l'ONSS ne peut pas assurer le traitement synchrone des remarques liées aux enregistrements de présences. En effet, le traitement des présences implique des appels à d'autres web services, rendant dès lors le traitement des remarques asynchrones.

Par conséquent, les web services ClaO décrivent deux opérations distinctes :

- la création de présences, et
- la consultation de présences.

Déroulement classique

Création

- L'utilisateur doit utiliser l'endpoint de création afin de soumettre ses présences. (***/presenceRegistrations/registerInBulk***)
- Il reçoit ensuite l'un des 2 codes suivants :
    - **200 :** dans ce cas, la réponse comporte autant d'objets que d'enregistrements de présence créées par l'utilisateur. Pour chaque enregistrement, un objet associé présente deux propriétés mutuellement exclusives :
        - **createdPresenceRegistration :** cette propriété décrit l'enregistrement de présence qui a été créé. Il est à noter que la propriété 'remarks' est vide. En effet, comme mentionné précédemment, le traitement des remarques est asynchrone. Par conséquent, lors de la création de l'enregistrement de présence dans ClaO, les remarques ne sont pas encore calculées. Si cette propriété est nulle, cela indique qu'une erreur s'est produite lors de l'enregistrement de la présence. Cette erreur est décrite dans la propriété '**notCreatedPresenceRegistration**'.
        - **notCreatedPresenceRegistration** : si cette propriété n'est pas nulle, cela indique qu'une erreur s'est produite lors de l'enregistrement de la présence. Le problème est décrit dans les propriétés de cet objet :
            - '**presenceRegistrationSubmitted**' : comme son nom l'indique, cette propriété reprend l'enregistrement de présence telle qu'il a été soumis par l'utilisateur. Cette présence contient donc une (ou des) erreur(s), décrite(s) dans la propriété '**errorList**'
            - '**errorList**' : contient une liste d'erreurs décrivant les raisons pour lesquelles l'enregistrement n'a pas pu être créé. Il convient dès lors de corriger les erreurs mentionnées, et de soumettre à nouveau **uniquement les présences en erreur**.
    - **500** : une erreur non gérée a eu lieu, les enregistrements de présences n'ont pas été créés.

Consultation

L'utilisateur dispose de deux méthodes afin de consulter les enregistrements de présences :

- Sur la base d'un **ID**, afin d'obtenir les informations d'une présence en particulier (***/presenceRegistrations/{id}***)
- Sur la base de **critères de recherche**, afin de lister la ou les présences qui correspondent aux critères mentionnés (***/presenceRegistration/search***)

Un utilisateur utilisant les web services sera toujours associé à l'employeur propriétaire du certificat. Dès lors, la consultation des enregistrements de présences sera restreinte aux enregistrements pour ce dit employeur, ou sa chaîne de sous-traitance.

Comme mentionné précédemment, lors de la création des présences, le champ **remarks** sera vide. Il est donc nécessaire d'attendre que CIaO traite les remarques avant d'essayer de les consulter.

Délais de traitement

Afin d'éviter de surcharger inutilement le web service de PresenceRegistration, il est essentiel de prendre en considération les informations suivantes :

- Chaque enregistrement de présence possède une propriété **status**. Cette propriété permet de suivre le statut du traitement des remarques. Les différentes valeurs suivantes sont possibles :
    - **REGISTERED** : la présence a été créée, mais elle n'a pas encore été traitée, ce qui signifie que la liste des remarques est vide mais susceptible d'évoluer.
    - **VALIDATED** : la présence a été traitée, et aucune remarque n'a été générée. La liste des remarques est donc vide, et les remarques ne seront plus calculées.
    - **CANCELLED** : la présence a été annulée, et par conséquent, ses remarques ne seront plus calculées.
    - **FAILED** : la présence a été traitée et des remarques ont été relevées. Cette liste de remarques est susceptible d'évoluer dans le futur (voir point 'batch'). Attention : l'enregistrement de présence est tout de même enregistré. Vous devez dans ce cas examiner les remarques et examiner si il est possible de les corriger.

Statut

Afin d'effectuer des vérifications sur le statut de la présence, il faut regarder à la propriété **code** de l'objet **status**. La **date** permet quant à elle de savoir quand le statut a été changé pour la dernière fois :

```
...
"status": {
```

```
    "code": "registered",
    "date": "2024-02-26T15:29:22.312422275+01:00"
},
```


Que signifie exactement la validité (validity) 'Failed' ?

La validité 'Failed' signifie que l'enregistrement de présence a correctement été reçu par l'ONSS, et qu'il a bien été enregistré dans notre base de données.

Cependant, l'enregistrement a été contrôlé et des remarques ont été constatées qui doivent attirer votre attention. Les remarques se trouvent dans la propriété '**remarks**'.

- Il peut s'agir d'un problème avec la déclaration Dimona. Dans ce cas, le problème doit être réglé dans le service en ligne 'Dimona'. Lorsque ce problème est résolu dans Dimona, il suffit d'attendre pour que cela se répercute dans ClaO (voir le point suivant, 'Batch').
- Il peut s'agir d'un problème avec la déclaration de travaux. Cela se résout dans le service en ligne 'Déclaration de travaux'. Par exemple, la remarque '12. Pas de DDT pour le BCE'. Si vous êtes entrepreneur-déclarant, cela signifie qu'un travailleur d'une entreprise non déclarée dans Déclaration de travaux (DDT) a travaillé pour votre déclaration. Vous devez dans ce cas, corriger la déclaration de travaux au plus vite, ou avertir l'ONSS.
- Il peut aussi s'agir de problème d'utilisation de Check In and Out at Work par vos travailleurs. Par exemple, la remarque '21. Enregistrement OUT manquant', signifie que nous avons reçu deux pointages 'IN' d'affilé pour ce travailleur.

Batch

Un système de traitement par lots (**batch**) s'exécute quotidiennement et recalcule les remarques des présences ayant le statut **REGISTERED** ou **FAILED.** Chaque jour, le batch recalculera toutes les remarques des présences pour les dates suivantes :

Soit D étant la date à laquelle le batch tourne :

- **D - 1** (les remarques de la veille)
- **D - 7** (les remarques créées il y a 1 semaine)
- **M - 1** (les remarques créées il y a 1 mois)
- **M - 3** (les remarques créées il y a 3 mois)

Il est donc inutile d'effectuer des appels chaque minute pour un enregistrement de présence créé il y a, par exemple, 5 jours. En effet, ces remarques ne seront pas re-traitées avant le prochain batch, qui sera exécuté 2 jours plus tard : le batch '**D - 7**'.

Bien que 95 % des enregistrement de présences soient traitées dans un délai de 10 secondes (en fonction du trafic), nous recommandons les maxima suivants en termes d'appels pour obtenir les remarques des présences nouvellement créées :

Temps après la création de la présence	Fréquence d'appel pour récupérer les remarques
de 0 à 1 minute	Un seul appel toutes les 5 secondes tant que la validité est à **PENDING**. En effet, si sa validité (validity) est à : • **FAILED** : les remarques n'évolueront plus avant J + 1, car elles ont été traitées. • **VALIDATED** : les remarques n'évolueront plus car la présence a été considérée comme valide.
à jour J + 1	Un appel unique. Les remarques ne seront plus traitées avant le prochain batch (à jour J + 7), et uniquement si la validité de la présence est **FAILED**.
à jour J + 7	Un appel unique. Les remarques ne seront plus traitées avant le prochain batch (à jour J + 30), et uniquement si la validité de la présence est **FAILED**.
à jour M + 1	Un appel unique. Les remarques ne seront plus traitées avant le prochain batch (à jour M + 1), et uniquement si la validité de la présence est **FAILED**.
à jour M + 3	Un appel unique. Les remarques ne seront plus traitées, et uniquement si la validité de la présence est **FAILED**.

Documentation technique

RegisterInBulk

Description

La méthode **registerInBulk** permet de créer une ou plusieurs présences (jusqu'à 200 par requête). Ces présences seront ensuite traitées de manières asynchrons par CIaO.

Requête

```
HTTP POST /presenceRegistrations/registerInBulk
```

Body

Le body de la requête doit contenir une liste d'objets ‘presence’, semblable à celui-ci :

```
{
   "items": [
       {...},
       {...}
   ]
}
```

Chaque objet doit correspondre à la description ci-dessous :

Nom de la propriété	Type de valeur	Obligatoire	Description
registrationDate	string <date-time>	Oui	La date à laquelle le IN ou le OUT a eu lieu, au **format ISO 8601 standard** : **YYYY-MM-DD*T*HH:MM:SS*Z*** • **YYYY-MM-DD** pour la date • **HH:MM:SS** pour l'heure • **Z** pour représenter la zone ou le fuseau horaire dans lequel la date est donnée
ssin	string <^\d{11}$>	Oui	Le NISS du travailleur qui a fait le IN ou le OUT

type	string ["IN", "OUT"]	Oui	Le type de présence : • IN : début de travail ou fin de pause • OUT : début de pause ou fin de travail
employer	Objet	Oui	L'employeur pour lequel la présence a été enregistrée. Cet objet doit obligatoirement contenir une et une seule de ces 2 propriétés: • **enterpriseNumber** : **string <^[0|1]\d{9}$>** : le numéro BCE de l'employeur (employeur belge) • **foreignVatNumber** : **string <**longueur maximale 255**>** : le numéro de TVA de l'employeur (employeur étranger)

placeOfWork	Objet	Oui	Le lieu de travail où la présence a été enregistrée. Cet objet doit obligatoirement contenir une et une seule de ces 2 propriétés :
			• **coordinates** : **objet** : les coordonnées du lieu de prestation sur lequel le travailleur enregistre sa présence (au format **WGS84**)
			  ○ **longitude** : **number** : la longitude de la coordonnées (X)
			  ○ **latitude** : **number** : la latitude de la coordonnées (Y)
			• **description** : **string <**longueur maximale 255**>** : champ libre pour décrit le lieu de prestation

contractualRelationshipReference	**string <^[A-HJ-NP-Z0-9]{13}$>**	Oui	La référence DDT du contrat entre le client/donneur d'ordre et le déclarant

Réponse

Pour chaque présence envoyée dans la requête, un objet associé est présent dans la réponse :

```
{
   "items": [
        {...},
        {...}
   ]
}
```

Chaque objet correspond à la description si dessous :

Nom de la propriété	Type de valeur	Obligatoire	Description
createdPresenceRegistration	Objet	Non	• Si une erreur s'est produite, cette propriété est nulle • Si l'enregistrement de présence a bien été enregistré, cette propriété contient la présence enregistrée (avec, donc, son *id* unique)
notCreatedPresenceRegistration	Objet	Non	• Si une erreur s'est produite, cet objet possède 2 propriétés : ○ **presenceRegistrationSubmitted** : qui décrit l'enregistrement de présence telle que soumis par l'utilisateur (qu'il convient donc de corriger) ○ **errorList** : une liste d'erreur, expliquant pourquoi l'enregistrement de présence n'a pas été enregistré. Chaque erreur possède 2 propriétés: ▪ **errorCode** : un code technique, unique à chaque erreur ▪ **errorDescription** : une description textuelle de l'erreur

Une présence ainsi créée possède l'ensemble des propriétés décrite dans le endpoint '**Read by id**'.

Exemple

dddd

Requête

Voici un exemple de requête contenant deux enregistrements de présence. Le premier est valide, le second ne l'est pas.

```
HTTP POST /presenceRegistrations/registerInBulk

{
  "items": [
    {
      "registrationDate": "2019-08-28T14:15:22Z",
      "ssin": "22663312345",
      "type": "in",
      "employer": {
        "enterpriseNumber": "450905686"
      },
      "placeOfWork": {
        "coordinates": {
          "longitude": 25.485606,
          "latitude": 20.673302
        }
      },
      "contractualRelationshipReference": "1Y1ZZZZZZZZZZ"
    },
    {
      "registrationDate": "2019-08-28T14:15:22Z",
      "ssin": "22663312345",
      "type": "in",
      "employer": {
        "enterpriseNumber": "4509056866666"
      },
      "placeOfWork": {
        "coordinates": {
          "longitude": 25.485606,
          "latitude": 20.673302
        }
      },
      "contractualRelationshipReference": "1Y1ZZZZZZZZZZ"
    }
  ]
}
```

Réponse

```
[
  {
    "createdPresenceRegistration": {
      "id": 17611,
      "registrationDate": "2019-08-28T15:15:22+02:00",
      "ssin": "22663312345",
      "type": "in",
```

```
      "employer": {
        "enterpriseNumber": "450905686",
        "foreignVatNumber": null
      },
      "placeOfWork": {
        "coordinates": {
          "longitude": 25.485606,
          "latitude": 20.673302
        },
        "description": null
      },
      "contractualRelationshipReference": "1Y1ZZZZZZZZZZ",
      "activity": "cleaning",
      "channel": "ws",
      "customReference": null,
      "status": {
        "code": "registered",
        "date": "2024-02-27T07:00:33.460412229+01:00"
      },
      "validity": "pending",
      "remarks": []
    },
    "notCreatedPresenceRegistration": null
  },
  {
    "createdPresenceRegistration": null,
    "notCreatedPresenceRegistration": {
      "presenceRegistrationSubmitted": {
        "id": null,
        "registrationDate": "2019-08-28T14:15:22Z",
        "ssin": "22663312345",
        "type": "in",
        "employer": {
          "enterpriseNumber": "4509056866666",
          "foreignVatNumber": null
        },
        "placeOfWork": {
          "coordinates": {
            "longitude": 25.485606,
            "latitude": 20.673302
          },
          "description": null
        },
        "contractualRelationshipReference": "1Y1ZZZZZZZZZZ",
        "activity": null,
        "channel": null,
        "customReference": null,
        "status": null,
        "remarks": []
      },
      "errorList": [
        {
          "errorCode": "error.presence-registration.creation.enterprise-
number",
          "errorDescription": "enterprise number is not valid"
        },
        {
```

```
          "errorCode": "error.presence-registration.creation.contractual-
relationship-reference",
          "errorDescription": "contractual relationship reference is not valid"
        }
      ]
    }
  }
]
```

Explications

- Le premier enregistrement de présence est tout à fait valide.
  - Dans la réponse, l'objet correspondant possède donc bien une propriété **createdPresenceRegistration** qui décrit la présence créée.
  - Dans la réponse, la propriété **notCreatedPresenceRegistration** est nulle, car il n'y a pas eu d'erreur.
- Le second enregistrement de présence n'est pas valide.
  - En effet, son NISS (**SSIN**) n'est pas valide.
  - Le numéro de d'entreprise de l'employeur n'est pas valide.
  - Ainsi, l'objet correspondant dans la réponse possède une propriété **createdPresenceRegistration** nulle.
  - L'objet possède également une propriété **notCreatedPresenceRegistration**, qui décrit:
    - **presenceRegistrationSubmitted** : la présence soumise, possédant donc une (ou des) erreur(s).
    - **errorList** : la liste des erreurs en question.

Read by id

Description

Cette méthode permet d'obtenir les informations relatives à une présence particulière, sur base de son ID technique.

Requête

Où **id** est l'ID de la présence dont vous voulez obtenir les informations :

```
HTTP GET /presenceRegistrations/{id}
```

Réponse

Cette méthode peut répondre de différentes façons :

Code de réponse	Interprétation
200	La présence a bien été trouvée.
404	• l'**id** mentionné dans la requête ne correspond à aucune présence. OU • la présence existe, mais elle n'est pas liée à l'employeur (ou sa chaîne de sous traitance) lié au token utilisé lors de la requête.

Dans le cas d'un code 200, la réponse contient donc un objet comme tel :

Nom de la propriété	Type de valeur	Description
id	**nombre**	ID technique unique de l'enregistrement de présence (à utiliser dans le endpoint de consultation)
registrationDate	**string <date-time>**	La date à laquelle le IN ou le OUT a eu lieu
ssin	**string <^\d{11}$>**	Le NISS du travailleur qui a fait le IN ou le OUT
worker	**Object**	Il s'agit du travailleur. Il contient ces 2 propriétés : • givenName : **string** : le prénom • familyName : **string** : le nom de famille
type	**string ["IN", "OUT"]**	Le type de présence : • IN • OUT

employer	Objet	L'employeur pour lequel l'enregistrement de présence a été enregistré. Cet objet doit obligatoirement contenir une et une seule de ces 2 propriétés : • **enterpriseNumber** : **string <^[0|1]\d{9}$>** : le numéro BCE de l'employeur (employeur belge) • **foreignVatNumber** : **string <**longueur maximale 255**>** : le numéro de TVA de l'employeur (employeur étranger)

placeOfWork	Objet	L'endroit où la présence a été enregistrée. Cet objet doit obligatoirement contenir une et une seule de ces 2 propriétés :
		
		• **coordinates** : objet : les coordonnées du lieu de prestation sur lequel le travailleur enregistre sa présence (au format **WGS84**)
		  ○ **longitude** : **number** : la longitude de la coordonnées (X)
		  ○ **latitude** : **number** : la latitude de la coordonnées (Y)
		• **description** : **string <**longueur maximale 255**>** : champ libre qui décrit le lieu de prestation

contractualRelationshipReference	**string <^[A-HJ-NP-Z0-9]{13}$>**	La référence DDT du contrat entre le client/donneur d'ordre et le déclarant
activity	**string**	Décrit le type d'activité réalisée par le travailleur
channel	**string**	Le canal via lequel la présence a été enregistrée : • **MOBILE_URL** • **MOBILE_MANUAL** • **WS** • **WEB_APP**
customReference	**string**	Référence personnalisable pour éventuellement faciliter l'intégration (cela permet de garder un lien entre l'ID technique de l'application source par exemple)

status	Objet	Le statut de la présence. Cet objet possède 2 propriétés:
		• **Code** : le code du statut actuel de la présence:
		  ○ **REGISTERED**
		  ○ **EDITED**
		  ○ **CANCELLED**
		• **Date** : la date à laquelle le statut a été changé la dernière fois

validity	Object	La validité de la présence: • **pending** • **failed** • **validated**

remarks	Liste d'objet 'remarque'	Détient une liste de toutes les remarques calculées pour cette présence. Un objet remarque possède les propriétés suivantes:
		• **Code** : le code unique de la remarque
		• **Labels** :
		  ○ **nl** : le nom de la remarque en néerlandais
		  ○ **fr** : le nom de la remarque en français
		  ○ **de** : le nom de la remarque en allemand
		  ○ **en** : le nom de la remarque en anglais

Exemple

Requête

```
HTTP GET /presenceRegistrations/17053
```

Réponse

```
{
      "id": 17053,
      "registrationDate": "2024-01-30T13:58:53.774+01:00",
      "ssin": "22663312345",
      "worker": {
         "givenName": "John",
         "givenName": "Doe"

      "type": "in",
```

```
      "employer": {
        "enterpriseNumber": "70010100188",
        "foreignVatNumber": null
      },
      "placeOfWork": {
        "coordinates": {
          "longitude": 4.348314,
          "latitude": 50.839552
        },
        "description": null
      },
      "contractualRelationshipReference": "1Y1ZZZZZZZZZZ",
      "activity": "cleaning",
      "channel": "mobile_manual",
      "customReference": null,
      "status": {
        "code": "failed",
        "date": "2024-01-30T13:58:59.930499+01:00"
      },
      "validity": "failed",
      "remarks": [
        {
          "code": "ciao_26",
          "labels": {
            "nl": "Gps-coördinaten niet gevonden in AVW",
            "fr": "Coordonnées GPS introuvables pour la DDT",
            "de": "GPS-Koordinaten für Arbeitsmeldung nicht gefunden",
            "en": "GPS coordinates not found for DOW"
          }
        },
        {
          "code": "ciao_34",
          "labels": {
            "nl": "Registratie geweigerd door CAW, controleer de gegevens",
            "fr": "Enregistrement refusé par CAW, veuillez vérifier les
données",
            "de": "Registrierung von CAW abgelehnt, bitte überprüfen Sie die
Daten",
            "en": "Registration refused by CAW, please check data"
          }
        }
      ]
}
```

Search

Description

Cette méthode permet de rechercher des présences correspondant à une liste de critères. En réponse, elle fournit une liste paginée des enregistrements de présences qui satisfont ces critères.

Une entreprise peut faire des recherches pour ses propres travailleurs ou ses sous-traitants.

Vous êtes éditeur de software ? Dans ce cas, vous devrez utiliser le certificat de votre client pour récupérer les enregistrements de présence. En effet, votre certificat permet la création d'enregistrements, mais pas la lecture de ceux-ci.

Requête

```
HTTP POST /presenceRegistrations/search
```

Body

Le **body** de la requête doit contenir un objet contenant une propriété **criteria** :

```
{
    "criteria": {
        ...
    }
}
```

Le corps de la requête doit répondre à la description ci-dessous :

Nom de la propriété	Type de valeur	Obligatoire	Valeur par défaut	Description

criteria	objet	Oui		Il s'agit des critères de recherche. Ces critères possèdent exactement les mêmes propriétés que l'objet présence décrit au point **Read by id** auxquelles s'ajoutent la propriété obligatoire suivante:
				
				• **registrationDate** :
				  ○ **startDate** : string <date-time>: la date de début sur base de laquelle filtrer les enregistrement de présences
				  ○ **endDate** : string <date-time>: la date de fin sur base de laquelle filtrer les enregistrements de présences
				
				Les enregistrement de présences qui résulteront de recherche possèderont donc une date d'enregistrement comprise entre registrationDate.startDate et registrationDate.endDate

sort	objet	Non	`{ "direction": "desc", "ignoreCase": false, "property": "registrationDate" }`	Un objet permettant de décrire la façon dont les résultats sont triés: • **direction** : ["ASC", "DESC"] pour ascendant ou descendant • **ignoreCase** : pour préciser si la recherche doit tenir compte de la casse ou pas • **property** : le nom de la propriété sur laquelle faire le tri

Autres paramètres

La requête peut contenir les paramètres de requête suivants :

Nom de la propriété	Type de valeur	Obligatoire	Valeur par défaut	Description
page	nombre	Non	1	La page à laquelle vous voulez accéder
pageSize	nombre	Non	50	Le nombre d'éléments par page

Réponse

Code de réponse	Interprétation
200	La recherche s'est bien déroulée, les résultats sont présents dans la réponse
500	Une erreur est survenue, vérifiez que les critères de recherche sont bien correctement formatés

Dans le cas d'un code 200, la recherche renvoie un objet correspondant à la description suivante :

Nom de la propriété	Type de valeur	Description
items	Liste d'objets	La liste des présences répondant aux critères de recherche. Les présences sont au format décrit au point **Read by ID**
first	string	Le lien vers la première page des résultats de la recherche
last	string	Le lien vers la dernière page des résultats de la recherche
prev	string	Le lien vers la page précédente des résultats de la recherche
next	string	Le lien vers la page suivante des résultats de la recherche
page	nombre	Le numéro de la page actuelle des résultats de recherche
pageSize	nombre	La taille de la page actuelle des résultats de recherche

sort	objet	Un objet permettant de décrire la façon dont les résultats sont triés: • **direction** : ["ASC", "DESC"] pour ascendant ou descendant • **ignoreCase** : pour préciser si la recherche doit tenir compte de la casse ou pas • **property** : le nom de la propriété sur laquelle faire le tri
total	nombre	Le nombre total d'éléments résultant de cette recherche
totalPages	nombre	Le nombre total de pages résultant de cette recherche

Exemple

Requête

```
HTTP POST /presenceRegistrations/search

{
    "criteria": {
        "registrationDate": {
            "startDate": "2024-01-30T10:12:52+01:00",
            "endDate": "2024-02-15T10:12:54+01:00"
        },
        "type": "in"
    }
}
```

Réponse

```
{
  "items": [
            {
            "id": 17053,
            ...
        },
        {
            "id": 17054,
            ...
        }
   ],
  "first":
"/REST/presenceRegistration/v1/presenceRegistrations/search?page=1&pageSize=50"
,
  "last":
"/REST/presenceRegistration/v1/presenceRegistrations/search?page=2&pageSize=50"
,
  "prev": null,
```

```
  "next":
"/REST/presenceRegistration/v1/presenceRegistrations/search?page=2&pageSize=50"
,
  "page": 1,
  "pageSize": 50,
  "sort": {
    "direction": "desc",
    "ignoreCase": false,
    "property": "registrationDate"
  },
  "total": 52,
  "totalPages": 2
}
```


Explications

Cette requête liste l'ensemble des enregistrement de présences, qui :

- possèdent une date d'enregistrement comprise entre le 30/01/2024 à 10:12 et le 15/02/2024 à 12:54 (champ '**registrationDate**' dans la requête)
- sont de type 'in' (champ '**type**' dans la requête)

Un critère est ajouté de manière transparente à la requête. Ces présences ont toutes été enregistrées pour l'employeur lié au token utilisé dans la requête.