ATELIER SAGE

Intacct API REST

Mohammed MEZOUGH

13 avril 2026

Vue d'ensemble

Sage Intacct propose une **API REST** pour connecter vos applications à ses **données**, **fonctions** et **flux de travail**.

Vos appels passent par **HTTP**, les payloads sont en **JSON**, et l'accès est protégé par **OAuth 2.0**.

CE QU'ON VERRA AUJOURD'HUI :

- **Démarrage et accès** · Quick Start, organisation développeurs, Sender ID
- **Authentification** · Client Credentials & Authorization Code
- **Modèle et découverte** · Service Model, champs, ressources personnalisées
- **Extraction et export** · Query, filtres, pagination, export
- **CRUD sur objets** · GET, POST, PATCH, DELETE
- **Opérations avancées** · Batch, Bulk, Composite, requêtes asynchrones
- **Extensions** · Conversions, dépôt GitHub, références

Sommaire

1

Démarrage et accès

Quick Start · Portail développeur · Société Intacct

Démarrage et accès · Portail développeur

1. Vérifier la licence Web Services : Sender ID et Sender Password sont requis.
2. Ouvrir developer.sage.com/intacct et cliquer sur **View Console**.
3. Se connecter avec un **Sage ID**, créer une **organisation développeurs**, puis s'inscrire au programme **Sage Intacct REST API**.
4. **Applications → New Application** — API **Sage Intacct**, créer l'application.
5. Configurer l'application — trois champs utiles à l'atelier : **Redirect URI** (ex. `https://company.local` si pas d'URL de callback réelle), mot de passe **licence Web Services** (Sender ID password), **Client Scope** (Production ou Non-production — non modifiable après création).
6. À l'issue de la création / configuration : noter le **Client ID** et le **Client Secret** (générés par le portail).

À conserver : Client ID · Client Secret · scope client.

Doc : Get started

> Atelier : configuration portail en direct (écran partagé).

Démarrage et accès · Société Intacct

Le portail fournit l'application OAuth — dans la **société**, on active l'API et on autorise *qui* peut l'appeler (prérequis du token ch. 2).

1. **Société · Admin · Abonnements** — activer **Services Web** (sinon pas d'API REST).
2. **Société · Configuration · Société · Sécurité · Autorisation des services web** — **Ajouter** le **Sender ID** (même expéditeur qu'au portail).
3. **Société · Admin · Utilisateurs des services web** — créer le compte technique (ID, e-mail — pas un utilisateur « métier » UI).
4. **Société · Configuration · Société · Sécurité · Applications clients autorisées** — lier **ID client** (Client ID) et **utilisateur Web Services** (associe l'application du portail au compte technique — sans ce lien, le flux **Client Credentials** ne peut pas aboutir).

À retenir : copier le Client ID et l'ID utilisateur WS **à l'identique** (sensible à la casse) — une variante empêche le token même si le secret est correct.

> Atelier : même parcours Intacct en direct (écran partagé).

2

Authentification

OAuth 2.0 · Client Credentials & Authorization Code

Authentification · Client Credentials

Ce flux sert aux accès **machine à machine**, sans interaction utilisateur.

OBTENIR UN JETON

Envoyer une requête **POST** vers `oauth2/token` avec un corps **form-urlencoded**.

1. Renseigner **grant_type** avec la valeur `client_credentials`.
2. Ajouter le **client_id** et le **client_secret** de l'application.
3. Indiquer **username** au format `webservice@company` ou `webservice@company|entity`.

À retenir : l'ID client doit être lié à un utilisateur Web Services (voir ch. 1). La réponse contient un **access_token** et un **refresh_token** (JWT) ; la durée est indiquée par **expires_in** (souvent 21600 s, soit 6 h).

Note : le suffixe `|entity` est optionnel. Vous choisissez le périmètre du jeton à la demande : top level ou entité uniquement.

Doc : OAuth 2.0

Authentification · Client Credentials · Exemple

Dans Postman, exécuter la requête **Get Token (Client Credentials)**.

`POST /oauth2/token`

CORPS

```
grant_type=client_credentials
client_id=xxx.app.sage.com
client_secret=••••••
username=webservice@company|entity
```

`200 OK`

RÉPONSE

```
{
  "access_token": "eyJ...",
  "refresh_token": "eyJ...",
  "token_type": "Bearer",
  "expires_in": 21600
}
```

Note : omettez `|entity` dans **username** pour un jeton top level. Corps en **form-urlencoded** (ou JSON selon la collection Postman).

Exemple Node : `client-credentials.mjs`

Authentification · Authorization Code

Ce flux sert lorsqu'un **utilisateur réel** se connecte via votre application (navigateur, mobile). Les appels API sont exécutés **à son nom**, et non via un compte Web Services.

FLUX EN 2 ÉTAPES

1. Dans le **navigateur**, rediriger l'utilisateur vers **GET** `oauth2/authorize` avec les paramètres requis en query string. Sage Intacct renvoie un **code** d'autorisation.
2. Côté **serveur**, échanger ce code via **POST** `oauth2/token` (corps form-urlencoded) pour récupérer les jetons.

À retenir : le code est à usage unique. Redirect URI en **https** (pas localhost) — **même valeur** à l'enregistrement de l'app, en étape 1 et en étape 2. Le **client_secret** ne doit jamais transiter côté navigateur.

Note : il n'y a pas de suffixe `|entity` dans ce flux. Le jeton est émis au niveau **top level**, ou sur l'**entité par défaut** de l'utilisateur si elle est définie dans Intacct.

Authentification · Authorization Code · Exemple · Étape 1

Étape navigateur : construire l'URL d'autorisation avec les paramètres en query string.

```
GET /oauth2/authorize?
response_type=code&client_id=xxx.app.sage.com&redirect_
uri=https://app.example/callback&scope=offline_access
```

302 Redirect

RÉPONSE · CALLBACK

```
https://app.example/callback
  ?code=AUTH_CODE_HERE
```

Note : Redirect URI = **https** (pas localhost) — même chaîne en étape 1 et 2. Après redirect, copier `code` depuis l'URL.

Exemple Node : `authorization-code.mjs` — les 2 étapes dans un fichier.

Authentification · Authorization Code · Exemple · Étape 2

Étape serveur : dans Postman, exécuter **Get Token (Authorization Code)** avec un corps form-urlencoded.

`POST /oauth2/token`

CORPS

```
grant_type=authorization_code
client_id=xxx.app.sage.com
client_secret=••••••
code=AUTH_CODE_HERE
redirect_uri=https://app.example/callback
```

`200 OK`

RÉPONSE

```
{
  "access_token": "eyJ...",
  "refresh_token": "eyJ...",
  "token_type": "Bearer",
  "expires_in": 21600
}
```

Démo : `authorization-code.mjs` (ouvrir URL → coller le code) ou Postman.

Authentification · Top level et entité

Dans une société **multi-entités**, le jeton fixe un contexte d'accès. Selon le flux utilisé, deux approches permettent de cibler les enregistrements d'une sous-entité.

CIBLER UNE SOUS-ENTITÉ

1. Conserver un jeton **top level** et ajouter le header `X-IA-API-Param-Entity` à chaque requête API, avec l'ID entité (propriété `company-config/entity.id`, ex. `CentralUS-35`).
2. Ou obtenir un jeton limité à l'entité dès le départ : suffixe `|entity` dans **username** (Client Credentials), ou paramètre **location_id** lors d'un **POST** `oauth2/token` avec `grant_type=refresh_token`.

À retenir : avec Client Credentials, vous choisissez le périmètre à l'émission du jeton. Avec Authorization Code, le jeton est top level ou entité par défaut de l'utilisateur — le header permet ensuite de cibler une autre entité si le jeton est top level.

Authentification · Top level et entité · Exemple

Jeton top level · cibler une entité par header ou par refresh.

`GET` **`/objects/vendor`**

HEADER · REQUÊTE API

```
Authorization: Bearer <top_level_token>
X-IA-API-Param-Entity: CentralUS-35
```

`POST` **`/oauth2/token`**

CORPS · REFRESH VERS JETON ENTITÉ

```
grant_type=refresh_token
client_id=xxx.app.sage.com
client_secret=••••••
refresh_token=eyJ...
location_id=CentralUS-35
```

Note : le refresh avec **location_id** renvoie un nouveau jeton utilisable pour les appels suivants, sans header entité à chaque requête.

Exemples Node : `entity-header.mjs` · `token-refresh-entity.mjs`

3

Modèle et découverte

Service Model · Champs et objets personnalisés

Modèle et découverte · Service Model

Le service **Model** retourne les **champs** et les **relations** d'une ressource — à consulter avant **Query** ou **CRUD**.

REQUÊTES TYPES

Tous les appels sont des **GET** vers `/services/core/model`, avec `Authorization: Bearer`.

1. `/services/core/model` — définition courte de toutes les ressources.
2. `?name=company-config/department` — modèle complet (`<application>/<ressource>`).
3. `?name=projects/task&version=v1` — modèle pour une version d'API.
4. `?version=v1&schema=true&type=workflow` — tous les modèles d'un type (object, service, workflow).

À retenir : objets custom `platform-apps/nsp::<nom>` · `?description=true` pour les descriptions · **schema** true/false selon liste ou ressource unique.

Doc : Model (OpenAPI)

Modèle et découverte · Service Model · Exemple

Avant Query ou CRUD sur les **fournisseurs**, on consulte le Model de `accounts-payable/vendor`.

```
GET /services/core/model?name=accounts-payable/vendor
```

200 OK

RÉPONSE · EXTRAIT (SANDBOX)


```
{
  "name": "accounts-payable/vendor",
  "fields": [
    { "id": "id", "label": "Vendor ID" },
    { "id": "name", "label": "Name" },
    { "id": "status", "label": "Status" },
    { "id": "billingType", "label": "Billing type" }
  ]
}
```


Note : le Model complet liste types, nullabilité et relations. Le **client** du fil facture expose des champs `nsp::` (slide suivante).

Exemple Node : `vendor.mjs`

Modèle et découverte · Champs personnalisés

Champs métier (UI) → préfixe `nsp::`. Exemple **client** Sage 100 sur `accounts-receivable/customer` — fil rouge **CL0642**.

1. Le **Model** liste `nsp::SAGE100_DB`, `nsp::SAGE100_ID`, …
2. Query / GET : inclure le préfixe dans **fields** ou le corps JSON.

POST `/services/core/query`

CORPS · CLIENT + CHAMPS SAGE 100

```
{
  "object": "accounts-receivable/customer",
  "fields": [
    "id", "name", "key",
    "nsp::SAGE100_DB", "nsp::SAGE100_ID"
  ],
  "filters": [{ "$eq": { "id": "CL0642" } }],
  "start": 1,
  "size": 1
}
```

200 OK

RÉPONSE · SANDBOX

```
{
  "ia::result": [{
    "id": "CL0642",
    "name": "Bague's en or modifié",
    "key": "181",
    "nsp::SAGE100_DB": "BIJOU",
    "nsp::SAGE100_ID": "BAGUES"
  }],
  "ia::meta": { "totalCount": 1 }
}
```

À retenir : même règle pour Export et Composite.

Exemple Node : `customer-nsp.mjs`

Modèle et découverte · Objets personnalisés

Objets custom → `platform-apps/nsp::{nom}` (ex. `platform-apps/nsp::partenaire` sur le sandbox).

1. **Model** — découvrir les champs (`id`, `name`, `montant`, …).
2. **Query** — même valeur pour `object` que pour le Model.
3. **CRUD** — `/objects/platform-apps/nsp::partenaire/{key}`

```
GET /services/core/model?name=platform-apps/nsp::partenaire&description=true
```

200 OK · Model (extrait)

CHAMPS

```
{
  "name": "platform-apps/nsp::partenaire",
  "fields": [
    { "id": "id" },
    { "id": "name" },
    { "id": "montant" },
    { "id": "key", "readOnly": true }
  ]
}
```

À retenir : champs `nsp::` sur objets *standard* (client) ≠ objets `platform-apps/nsp::` · **Query** : `POST /services/core/query` avec `"object": "platform-apps/nsp::partenaire"` (ch. 4) · droits requis sur l'objet custom.

Exemples Node : `custom-object-partenaire.mjs` · `query-partenaire.mjs`

4

Extraction et export

Query · Export

Extraction et export · Query

Le service **Query** retourne les objets filtrés d'une société Sage Intacct.

REQUÊTE PRINCIPALE

POST `/services/core/query` — corps JSON.

1. **object** — `accounts-payable/vendor` ou `platform-apps/nsp::<nom>`
2. **fields** — champs, relations (`vendor.id`), agrégats (`max:vendor.creditLimit`)
3. **filters** + **filterExpression** — `$eq`, `$in`, `$contains`, …

À retenir : consulter le **Model** pour les noms de champs valides.

Doc : Query service · Query (OpenAPI)

Extraction et export · Query · Pagination

Affiner et parcourir de gros volumes.

1. **orderBy** — `[{"id": "asc"}]`
2. **start** — premier enregistrement (ex. 1)
3. **size** — nombre de lignes, **maximum 4000**
4. **caseSensitiveComparison** — défaut `true`
5. **asOfDate**, **includePrivate**, **includeHierarchyFields** — cas avancés

À retenir : pour très gros volumes, paginer avec Query ou utiliser **Export** / **Bulk**.

Extraction et export · Query · Exemple

On filtre les **fournisseurs** actifs en open item — même logique que pour les factures plus loin.

`POST /services/core/query`

CORPS

```
{
  "object": "accounts-payable/vendor",
  "fields": ["id", "name", "status", "href"],
  "filters": [
    {"$eq": {"status": "active"}},
    {"$eq": {"billingType": "openItem"}}
  ],
  "filterExpression": "1 and 2",
  "orderBy": [{"id": "asc"}],
  "start": 1,
  "size": 100
}
```

`200 OK`

RÉPONSE · EXTRAIT

```
{
  "ia::result": [
    {
      "id": "FR0693",
      "name": "SAGE",
      "status": "active",
      "href": "/objects/accounts-payable/vendor/1",
      "billingType": "openItem"
    },
    {
      "id": "VND-001",
      "name": "Vendor 007 entity",
      "status": "active",
      "href": "/objects/accounts-payable/vendor/43",
      "billingType": "openItem"
    }
  ],
```

À retenir : réponse JSON paginée — ici 17 fournisseurs sur le sandbox.

Exemple Node : `vendors.mjs`

Extraction et export · Query · Factures (AR)

Même service Query sur le **fil facture** : champs liés via `customer.id` et `customer.name` (équivalent de `vendor.*` côté AP).

`POST /services/core/query`

CORPS

```
{
  "object": "accounts-receivable/invoice",
  "fields": [
    "id", "invoiceNumber",
    "customer.id", "customer.name",
    "invoiceDate", "dueDate",
    "totalTxnAmount", "state"
  ],
  "filters": [{ "$gt": { "totalTxnAmount": "0" } }],
  "orderBy": [{ "invoiceDate": "desc" }],
  "start": 1,
  "size": 100
}
```

`200 OK`

RÉPONSE · EXTRAIT (SANDBOX)

```
{
  "ia::result": [
    {
      "id": "69",
      "invoiceNumber": "bm",
      "customer.id": "CL0642",
      "customer.name": "Bague's en or modifié",
      "invoiceDate": "2026-05-22",
      "dueDate": "2026-07-05",
      "totalTxnAmount": "200.00"
    }
  ],
  "ia::meta": { "totalCount": 3, "start": 1, "pageSize": 100 }
}
```

Exemple Node : `invoices.mjs`

Extraction et export · Export

Le service **Export** produit un **fichier** à partir d'une requête filtrée.

FORMATS

POST `/services/core/export` — **fileType** + objet **query**.

1. `csv`, `pdf`, `word`, `xml`, `xlsx`
2. Même structure de filtrage que Query, imbriquée dans **query**

À retenir : réponse = fichier téléchargeable.

Doc : Export (OpenAPI) — options du `POST /services/core/export`

Extraction et export · Export · Exemple

Même filtres que Query, mais la réponse est un **fichier** (PDF, CSV, ...) — pratique pour l'utilisateur métier.

`POST /services/core/export`

CORPS

```
{
  "fileType": "pdf",
  "query": {
    "object": "accounts-payable/vendor",
    "fields": ["id", "name", "status", "href"],
    "filters": [
      {"$eq": {"status": "active"}},
      {"$eq": {"billingType": "openItem"}}
    ],
    "filterExpression": "1 and 2",
    "orderBy": [{"id": "asc"}]
  }
}
```

`200 OK`

RÉPONSE · MÉTADONNÉES (SANDBOX)

```
{
  "status": 200,
  "statusText": "OK",
  "contentType": "application/pdf",
  "sizeBytes": 8421,
  "_note": "PDF binaire non affiché"
}
```

À retenir : pas de tableau `ia::result` — fichier binaire (PDF).

Exemple Node : `vendors-pdf.mjs`

5

CRUD sur objets

GET · POST · PATCH · DELETE

CRUD · Lecture (GET)

Ressources : `/objects/{application}/{object}` et `/.../{key}` pour le détail.

1. **Liste** — références légères (id, key, href) + `ia::meta`
2. **Détail** — enregistrement complet par **key**
3. **Limitation** — la liste ne remplace pas **Query** (pas de filtres riches / pagination comme Query)

À retenir : exploration → GET liste ; extraction métier → Query · `Authorization: Bearer` sur chaque appel (ch. 2), non répété sur les exemples JSON.

Doc : OpenAPI — objet métier (ex. invoice, vendor)

CRUD · GET · Liste

La liste renvoie des références légères ; elle peut être vide avant la première création en sandbox.

```
GET /objects/accounts-receivable/invoice
```

200 OK

RÉPONSE · EXTRAIT (SANDBOX)


```
{
  "ia::result": [
    {"key": "63", "id": "63", "href": "/objects/accounts-receivable
  ],
  "ia::meta": {
    "totalCount": 1,
    "start": 1,
    "pageSize": 100
  }
}
```


Exemple Node : `get-invoices-list.mjs`

CRUD · GET · Détail

Avec la **key** de la liste, on charge la facture complète — client, montants et **lignes**.

`GET /objects/accounts-receivable/invoice/67`

`200 OK`

RÉPONSE · EXTRAIT (SANDBOX)


```
{
  "ia::result": {
    "key": "67",
    "invoiceDate": "2026-05-21",
    "dueDate": "2026-07-15",
    "referenceNumber": "PO-UPDATED-99",
    "description": "Modifié par Atelier — en-tête facture",
    "totalTxnAmount": "150.00",
    "customer": { "id": "CL0642", "name": "Bague's en or modifié" },
    "currency": { "baseCurrency": "EUR", "txnCurrency": "EUR" },
    "lines": [{
      "key": "137",
      "txnAmount": "150.00",
      "memo": "Ligne mise à jour — atelier",
      "glAccount": { "id": "701000", "name": "Ventes de produits f
      "dimensions": {
        "location": { "id": "DEMO_1", "name": "Entité 1" }
```


Note : la **key** de ligne (`137`) sert au PATCH sur `invoice-line`.

Exemple Node : `get-invoice-detail.mjs`

CRUD · Création (POST)

POST `/objects/{application}/{object}` — corps JSON avec champs obligatoires et objets liés.

1. Succès **201 Created**
2. Réponse `ia::result` : **id**, **key**, **href**

CRUD · POST · Exemple

On crée une facture AR pour **CL0642** — en-tête + une ligne, champs issus du Model atelier.

`POST /objects/accounts-receivable/invoice`

CORPS · EXTRAIT

```
{
  "customer": { "id": "CL0642" },
  "invoiceDate": "2026-05-21",
  "dueDate": "2026-06-20",
  "referenceNumber": "PO-ATELIER-001",
  "description": "Facture démo atelier REST API",
  "lines": [{
    "txnAmount": "100",
    "glAccount": { "id": "701000" },
    "memo": "Prestation atelier",
    "dimensions": {
      "customer": { "id": "CL0642" },
      "location": { "id": "DEMO_1" }
    }
  }]
}
```

`201 Created`

RÉPONSE (SANDBOX)

```
{
  "ia::result": {
    "key": "67",
    "id": "67",
    "href": "/objects/accounts-receivable/invoice/67"
  },
  "ia::meta": { "totalSuccess": 1, "totalError": 0 }
}
```

Model d'abord : `invoiceDate`, `dueDate`, `referenceNumber`, `description` — voir Model `accounts-receivable/invoice`.

Exemple Node : `post-invoice.mjs`

CRUD · Mise à jour (PATCH)

PATCH `/objects/.../{key}` — mise à jour **partielle** : champs absents = inchangés.

Succès : 200 OK. Champs modifiables : voir le **Model** · lignes de document : endpoints dédiés.

CRUD · PATCH · Exemple

On ne renvoie que ce qui change — ici référence, description et échéance de la facture 67.

`PATCH /objects/accounts-receivable/invoice/67`

CORPS

```
{
  "referenceNumber": "PO-UPDATED-99",
  "description": "Modifié par Atelier — en-tête facture",
  "dueDate": "2026-07-15"
}
```

`200 OK`

RÉPONSE (SANDBOX)

```
{
  "ia::result": {
    "key": "67",
    "id": "67",
    "href": "/objects/accounts-receivable/invoice/67"
  }
}
```

À retenir : pas besoin de renvoyer tout l'objet — seuls les champs du corps sont mis à jour.

Exemple Node : `patch-invoice.mjs`

CRUD · Suppression (DELETE)

DELETE `/objects/.../{key}` — pas de corps.

Succès : 204 No Content.

> Vérifier dépendances et règles métier avant suppression en production.

CRUD · PATCH · Ligne de facture

Une ligne se met à jour sur `/objects/accounts-receivable/invoice-line/{key}`, pas sur l'en-tête facture.

PATCH /objects/accounts-receivable/invoice-line/137

CORPS

```
{
  "txnAmount": "150.00",
  "memo": "Ligne mise à jour — atelier"
}
```

200 OK

RÉPONSE (SANDBOX)

```
{
  "ia::result": {
    "key": "137",
    "id": "137",
    "href": "/objects/accounts-receivable/invoice-line/137"
  }
}
```

Exemple Node : `patch-invoice-line.mjs`

CRUD · DELETE · Exemple

Nettoyage de la facture de démo — pas de corps, réponse 204 si tout va bien.

```
DELETE /objects/accounts-receivable/invoice/67
```

204 No Content

RÉPONSE

```
(corps vide)
```

Exemple Node : `delete-invoice.mjs`

6

Opérations avancées

Batch · Bulk · Composite · Asynchrone

Opérations avancées · Batch

Éviter des centaines d'appels identiques : un **tableau JSON**, un seul POST — tout est traité tout de suite (même type d'objet, max 500).

1. **GET** — `/objects/.../bill/194,195,310`
2. **DELETE** — clés séparées par virgules → **204**
3. **POST / PATCH** — corps = **tableau JSON** ; PATCH : **key** dans chaque objet

Header : `X-IA-API-Param-Transaction: true` pour atomicité (rollback si un échec).

Doc : Batch, bulk & composite

Opérations avancées · Batch · Exemple

Trois **fournisseurs** créés en une fois — utile pour des imports de taille moyenne.

`POST /objects/accounts-payable/vendor`

CORPS · TABLEAU

```
[
  { "id": "batchv1-…", "name": "Batch Vendor 1 …" },
  { "id": "batchv2-…", "name": "Batch Vendor 2 …" },
  { "id": "batchv3-…", "name": "Batch Vendor 3 …" }
]
```

`200 OK`

RÉPONSE (SANDBOX)

```
{
  "ia::result": [
    { "key": "48", "id": "batchv1-0521130212", "href": "/objects/ac
    { "key": "49", "id": "batchv2-0521130212", "href": "/objects/ac
    { "key": "50", "id": "batchv3-0521130212", "href": "/objects/ac
  ],
  "ia::meta": { "totalCount": 3, "totalSuccess": 3, "totalError": (
}
```

À retenir : `totalSuccess` / `totalError` dans `ia::meta` — ou **207** si succès et échecs mélangés.

Exemple Node : `batch-post-vendors.mjs`

Opérations avancées · Bulk · Soumission

Pour de **très gros volumes** : on dépose un fichier, Intacct traite en arrière-plan — on récupère un **jobId**, pas une attente bloquante.

POST `/services/bulk/job/create` — **multipart/form-data** (métadonnées + fichier JSON).

PARTIE IA::REQUESTBODY

```
{
  "objectName": "accounts-payable/vendor",
  "operation": "create",
  "jobFile": "file",
  "fileContentType": "json"
}
```

+ PARTIE FILE (EX. FOURNISSEURS)

```
[
  { "id": "bulkv…1", "name": "Bulk Vendor 1" },
  { "id": "bulkv…2", "name": "Bulk Vendor 2" }
]
```

MÊME SYNTAXE POUR CLIENTS AR

201 Created

RÉPONSE (SANDBOX)

```
{
  "ia::result": {
    "jobId": "95884b67-fd9e-42ba-af94-b39dee3d53f1",
    "statusURL": "/services/bulk/job/status?jobId=…"
  }
}
```

file = tableau JSON · champs minimaux selon le **Model** de l'objet.

Doc : Batch, bulk & composite — section Bulk

Exemple Node : `bulk-create-vendors.mjs`

Opérations avancées · Bulk · Statut

On interroge le job jusqu'à **completed**, puis on télécharge le fichier résultat (succès / erreurs par ligne).

GET `/services/bulk/job/status?jobId=…&download=true`


```
{
  "ia::result": {
    "jobId": "f426e9c9-130d-4a9f-9dae-3a3a2935f850",
    "status": "completed",
    "percentComplete": 100
  }
}
```


À retenir : distinct du `/services/core/async/job-status` (header `Prefer: respond-async`).

Exemple Node : `bulk-job-status.mjs`

Opérations avancées · Composite

Un seul appel HTTP pour **enchaîner** plusieurs opérations — la réponse d'une étape peut alimenter la suivante (ex. créer une facture puis la relire).

POST `/services/core/composite` — tableau de sous-requêtes, exécutées dans l'ordre.

1. Chaque entrée : **method**, **path**, **body** (POST/PATCH), **headers**, **resultReference**
2. Réponse : `ia::result` par sous-requête + `ia::meta` (totalSuccess, totalError)

Idempotency-Key recommandé pour éviter les doublons.

Doc : Batch, bulk & composite — section Composite

Opérations avancées · Composite · Exemple

Étape 1 : POST facture · étape 2 : GET la même facture avec la **key** renvoyée — sans second appel « à la main ».

`POST /services/core/composite`

CORPS · EXTRAIT

```
[
  {
    "method": "POST",
    "path": "/objects/accounts-receivable/invoice",
    "body": {
      "customer": { "id": "CL0642" },
      "invoiceDate": "2026-05-21",
      "dueDate": "2026-06-20",
      "lines": [{ "txnAmount": "100", "glAccount": { "id": "701000"
        "dimensions": { "customer": { "id": "CL0642" }, "location"
    },
    "resultReference": "newInvoice"
  },
  {
    "method": "GET",
    "path": "/objects/accounts-receivable/invoice/@{newInvoice.key}
  }
```

`200 OK`

RÉPONSE (SANDBOX)

```
{
  "ia::result": [
    { "ia::status": 201, "ia::result": { "key": "75", "id": "75" }
    { "ia::status": 200, "ia::result": {
        "key": "75", "referenceNumber": "PO-COMP-…",
        "customer": { "id": "CL0642", "name": "…" }
    }}
  ],
  "ia::meta": { "totalSuccess": 2, "totalError": 0 }
}
```

À retenir : `resultReference` + `@{newInvoice.key}` — on peut aussi enchaîner client → facture, si le tenant l'autorise.

Exemple Node : `composite-invoice.mjs`

Opérations avancées · Asynchrone

Quand l'opération est longue : on ne bloque pas la connexion — Intacct répond tout de suite **202** avec un **jobId** à suivre.

Header `Prefer: respond-async` sur la requête métier (POST, PATCH, …).

HEADERS

```
Prefer: respond-async
Authorization: Bearer …
```

202 Accepted

ACK · EXTRAIT

```
{
  "ia::result": {
    "jobId": "NjQ2NTc2MzAzMNl1Ul9qVmd6M2t4M12pPdEJya2Jt65Y2dBQUFBQ
    "state": "queued",
    "queuedDateTime": "2025-11-11T00:47:12Z",
    "href": "/services/core/async/job-status?jobId=…"
  },
  "ia::meta": { "totalCount": 1 }
}
```

Retry-After ~ 2 min · états : `queued` , `inTransit` , `delivered` , `dead` , … · Async URIs (ch. 1) pour webhooks.

Doc : Asynchronous requests

Exemple Node : `async-prefer.mjs`

Opérations avancées · Quel service choisir ?

Récapitulatif — le bon outil selon le besoin métier.

- **Query** — lire / filtrer · pagination `size` max **4000**
- **Batch** — ≤ **500** enregistrements, **même objet**, réponse immédiate (synchrone)
- **Bulk** — très gros volume · fichier JSON · job `jobId` + téléchargement résultat
- **Composite** — plusieurs opérations (GET/POST/PATCH mix) en **un seul HTTP** · `resultReference`
- **Async** — `Prefer: respond-async` · ACK **202** + job-status (≠ Bulk)

À retenir : Bulk `/services/bulk/job/status` ≠ core async `/services/core/async/job-status` — même idée de suivi, services différents.

> Volume / concurrence API → API volume & scaling — pas de détail en atelier.

7

Extensions et clôture

Conversions · Dépôt · Références

Clôture · Dépannage courant

Symptôme HTTP → première piste de vérification.

1. **401 Unauthorized** — jeton absent, expiré ou invalide → refaire `oauth2/token` (ch. 2)
2. **403 Forbidden** — utilisateur WS / droits objet / entité → ch. 1 (application autorisée) + permissions Intacct
3. **400 Bad Request** — corps JSON invalide, champ inconnu ou non modifiable → **Model d'abord** (ch. 3) ; lire `ia::error`

```
{
  "ia::result": {
    "ia::error": {
      "code": "invalidRequest",
      "message": "A POST request requires a payload",
      "errorId": "REST-1028"
    }
  },
  "ia::meta": { "totalError": 1 }
}
```

À retenir : en atelier, Postman affiche le corps d'erreur — lire `errorId` + `details`, ne pas deviner.

Extensions · Conversion de documents

Order Entry et Purchasing — **POST** sur la **définition cible** (pas le document source) ; lier la source via `sourceDocument` et les clés de ligne.

1. Endpoint = type **destination** : `/objects/order-entry/document::Sales Return` (ex. Sales Invoice → Sales Return)
2. Corps : **sourceDocument** en en-tête ; dans chaque ligne **sourceDocument** + **sourceDocumentLine** — de préférence par **key** (GET / Query)
3. Crée le nouveau document et **marque la source comme convertie**

À retenir : Model + dossiers Postman Order Entry / Purchasing pour les payloads complets ; clés source depuis GET ou Query.

Purchasing — même mécanisme : PO → `POST /objects/purchasing/document::Vendor Invoice` · Document conversions

Extensions · Conversion de documents · Exemple Order Entry

Sales Invoice → Sales Return — extrait doc Sage.

POST `/objects/order-entry/document::Sales Return`

CORPS · EXTRAIT

```
{
  "sourceDocument": { "key": "12920" },
  "lines": [{
    "sourceDocument": { "key": "12920" },
    "sourceDocumentLine": { "key": "14468" },
    "dimensions": { "item": { "id": "1" } },
    "unitQuantity": "1",
    "unitPrice": "1000"
  }]
}
```

201 `Created`

RÉPONSE · EXTRAIT

```
{
  "ia::result": {
    "key": "12922",
    "id": "12922",
    "href": "/objects/order-entry/document::Sales%20Return/12922"
  },
  "ia::meta": {
    "totalSuccess": 1,
    "totalError": 0
  }
}
```

À retenir : champs en-tête complets (customer, dates, devises…) — voir Document conversions.

Extensions · Dépôt et démos

Démo **Postman** et exemples Node (dépôt public).

- github.com/mmezough/intacct-rest-api-atelier — un script par concept
- Collection Postman officielle — dossiers par service
- JSON d'exemple déjà sur les slides · code Node via liens `blob/` GitHub

> Configurer `lib/config.mjs` (ou variables Postman) avant la démo live.

Références

- Exemples Node — intacct-rest-api-atelier
- Documentation REST — démarrage · XML ↔ REST
- OAuth 2.0
- Query service · Query OpenAPI
- Model (OpenAPI)
- Batch, bulk & composite
- Asynchronous requests
- Document conversions
- Volume / concurrence / scaling
- Référence OpenAPI
- Collection Postman · tutoriel
- Portail développeur

ATELIER SAGE

Merci

Questions · échanges

Exemples Node · Documentation