KodeMed

Guide d’intégration

Pour les développeurs et intégrateurs système

Version 2026.5.15.59502 | KodeMed AG

Vue d’ensemble de l’intégration

Deux voies d’intégration vers KodeMed

Option A : DLL COM

Intégration directe au SIH

- Charger la DLL via la variable KODEMED_DLL_PATH
- Appeler SendConfig() avec XML ou JSON
- Appeler DoCodingWithFormat() pour le flux complet avec UI
- Appeler DoCoding() pour le traitement headless
- Récupérer les résultats via GetResults()
- Compatible .NET, VB6, VBA, C#, Delphi

Option B : API REST / WebSocket

Applications tierces via CodingClient

- POST /api/v1/coding/session avec les données
- Le serveur transmet au CodingClient via WebSocket
- CodingClient ouvre l’UI de codage sur le poste de l’utilisateur
- Interroger le statut de session via endpoint GET
- Tout langage/plateforme (Java, Python, ...)
- Mises à jour en temps réel via WebSocket

Option A : Intégration DLL COM

Étape par étape pour les applications .NET / VB / C#

Exemple C#

```
var coding = new Coding();

// 1. Configure
coding.SendConfig(configXml);

// 2. Process with UI (auth + browser)
coding.DoCodingWithFormat(data,
FormatType.SpiGes);

// 3. Get results
string results = coding.GetResults();

// 4. Cleanup
coding.DisposeClient();
```

Exemple VB.NET / VBA

```
Dim km As Object
Set km = CreateObject("KodeMed.Coding")
km.SendConfig configXml
km.DoCodingWithFormat data, 0  ' SpiGes
result = km.GetResults()
km.DisposeClient
```

Résolution du chemin DLL

- Variable KODEMED_DLL_PATH (définie par CodingClient)
- COM : CreateObject("KodeMed.Coding")
- Réflexion : Assembly.LoadFile(dllPath)

Option B : Intégration API REST

Pour tout langage/plateforme

Flux des endpoints

Step	Method	Endpoint	Description
1	POST	/api/v1/coding/session	Create session with case data + format
2	-	(WebSocket push)	Server pushes CODING_SESSION_LAUNCH to CodingClient
3	GET	/api/v1/coding/session/{id}	Poll session status
4	GET	/api/v1/coding/session/{id}/result	Retrieve coding results

Exemple : Créer une session (curl)

```
curl -X POST https://server/api/v1/coding/session \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"data": "<SpiGes XML>", "format": "spiges", "instanceId": "my-app-1"}'

# Response: {"sessionId": "abc-123", "redirectUrl": "https://ui/spiges/session/abc-123/cases"}
```

Option C : Intégration WebSocket

Communication bidirectionnelle en temps réel

Types de messages WebSocket

Type	Direction	Description
CONNECTED	S -> C	Connection established
HEARTBEAT	S -> C	Keep-alive ping
HEARTBEAT_ACK	C -> S	Heartbeat response
CODING_SESSION_LAUNCH	S -> C	Launch coding session
SESSION_ACTION	C -> S	Apply/Discard from UI
SESSION_CLOSE	C -> S	DLL closes session
UI_SESSION_JOIN	C -> S	UI joins coding session

Connexion

```
// Connect
ws://server:8080/ws/dll

// With auth token
ws://server:8080/ws/dll
  ?token=<jwt>
  &instanceId=<id>

// Auto-reconnect: 60s
```

Format du payload (JSON)

```
{
  "type": "CODING_SESSION_LAUNCH",
  "instanceId": "dll-xyz",
  "payload": { "data": "...",
    "format": "spiges" }
}
```

Authentification

OAuth2 / OpenID Connect

Flux OAuth2 / OIDC

- Compatible avec : Keycloak, Auth0, Azure AD, Okta
- Flux Authorization Code avec PKCE
- La DLL ouvre un navigateur intégré pour la connexion
- Rafraîchissement auto des tokens (access + refresh)
- Langue utilisateur lue depuis le claim JWT « locale »
- Client IDs : kodemed-dll (DLL), kodemed-ui (navigateur)
- Realm : configurable (par défaut : kodemed)

Configuration

```
<OAuth2Url>
  https://sso.example.com/auth
</OAuth2Url>
<OAuth2Realm>kodemed</OAuth2Realm>
<OAuth2ClientId>kodemed-dll</OAuth2ClientId>
```

Cycle de vie des tokens

- Access token : courte durée (~5 min)
- Refresh token : longue durée (~30 min)
- Rafraîchissement automatique avant expiration
- Reconnexion WebSocket rafraîchit le token

Formats de données

SpiGes XML, OFS, Personnalisé

Structure XML SpiGes (primaire)

```
<?xml version="1.0" encoding="UTF-8"?>
<Unternehmen xmlns="...spiges-data/1.5"
  ent_id="100000001" version="1.5">
  <Standort burnr="10000001">
    <Fall fall_id="1">
      <Administratives
        abc_fall="A" alter="38"
        geschlecht="1"
        eintrittsdatum="2022061010"
        austrittsdatum="2022061209" />
      <Diagnose diagnose_id="1"
        diagnose_kode="K3530"
        diagnose_poa="3" />
      <Behandlung behandlung_id="1"
        behandlung_chop="4701"
        behandlung_beginn="202206110000" />
    </Fall>
  </Standort>
</Unternehmen>
```

Enum FormatType

Value	Format	Extensions
0	SpiGes	.xml
1	BFS	.dat, .xml
2	Custom	plugin-defined

Système de plugins

- Implémenter l’interface IFormatPlugin
- Découverte automatique depuis le répertoire plugins
- Transformer les données personnalisées en modèle SpiGes
- Intégré : JSON Clinical Plugin

Setup CodingClient

Application system tray portable

Installation (3 étapes)

- 1. Copier le dossier KodeMed à n’importe quel emplacement
- 2. Exécuter KodeMed.CodingClient.exe
- 3. Auto-setup : variables d’env + enregistrement DLL COM

Variables d’environnement (définies automatiquement)

Variable	Value
KODEMED_HOME	Working directory
KODEMED_DLL_PATH	Full path to KodeMed.dll
KODEMED_EXE_PATH	Full path to CodingClient.exe

Compatibilité Citrix / VDI

- EXE autonome : aucun runtime .NET requis
- Niveau utilisateur uniquement : toutes les écritures dans HKCU
- Aucun droit administrateur requis
- Enregistrement COM via HKCU\Software\Classes
- Config : à côté de l’EXE ou %KODEMED_HOME%
- VDI non persistant : re-setup à chaque connexion

Menu tray (multilingue : DE/FR/IT/EN)

- Connexion / Déconnexion (OAuth2)
- Ouvrir session depuis fichier (SpiGes / OFS / Personnalisé)
- Langue (DE/FR/IT/EN)
- Paramètres serveur, Démarrage auto, Setup / Réparation

Référence API rapide

Méthodes DLL, Endpoints REST, Messages WebSocket

Méthodes DLL (IKodeMed)

Method	Description
SendConfig(string)	Load XML/JSON configuration
DoCoding(string, FormatType)	Process data (headless, no UI)
DoCodingWithFormat(string, FormatType)	Process with UI (auth + browser)
GetResults()	Get results (XML or JSON)
DisposeClient()	Release all resources

Endpoints REST

Method	Endpoint	Action
POST	/api/v1/coding/session	Create session
GET	/api/v1/coding/session/{id}	Get status
GET	/api/v1/coding/session/{id}/result	Get results
POST	/api/v1/coding/session/{id}/complete	Complete session
POST	/api/v1/coding/session/{id}/cancel	Cancel session
GET	/api/v1/config	Get server config

Messages WebSocket (ws://server/ws/dll)

Message	Direction	Payload	Description
CODING_SESSION_LAUNCH	Server -> Client	data, format, sessionId	Launch coding UI for new session
SESSION_ACTION	Client -> Server	sessionId, action	User applied or discarded changes
SESSION_CLOSE	Client -> Server	sessionId	DLL/WebView2 closed the session

Webhook côté DLL

Émis par CodingClient — l’ancien flux in-process

Fonctionnement

- Le codeur termine la session → CodingClient envoie du JSON en POST
- Fire-and-forget : ne bloque pas le codeur
- Retry avec backoff exponentiel (1s→2s→4s... max 60s)
- Pas de retry sur 4xx (sauf 429 Too Many Requests)

Configuration serveur (variables d’env)

Variable	Default
KODEMED_HOOK_ENABLED	false
KODEMED_HOOK_URL	(empty)
KODEMED_HOOK_INCLUDE_RESULT_DATA	false (opt-in)
KODEMED_HOOK_INCLUDE_GROUPER_RESULTS	false (opt-in)
KODEMED_HOOK_EVENTS	applied

Configuration client (auth locale)

```
// kodemed-client-config.json
{
  "hook": {
    "authType": "bearer",
    "authToken": "sk-hospital-key"
  }
}
```

Minimisation des données (RGPD/nLPD)

- includeResultData par défaut false (opt-in)
- includeGrouperResults par défaut false (opt-in)
- Identifiants d’auth jamais envoyés par le serveur
- Protection SSRF sur les URL de webhook

Contrat Webhook HIS REST

Émission côté serveur sur case.coded / case.discarded

Déclencheur

- Le codeur clique Apply → event_type = case.coded
- Le codeur clique Discard → event_type = case.discarded
- Par tenant : IntegrationProfile.webhookUrl + webhookEventTypes
- Émis par le Serveur (PAS la DLL) — pour les partenaires sans DLL

Idempotence

- En-tête Idempotency-Key : <session_id>:<event_type>
- Stable entre les retries — déduplication côté SIH via cet en-tête
- Livraison « at-least-once » : prévoir les doublons côté récepteur

Payload (HisWebhookPayload)

```
{
  "event_type":  "case.coded",
  "occurred_at": "2026-05-12T21:45:12Z",
  "session_id":  "550e8400-…-446655440000",
  "instance_id": "his-instance-12345",
  "spiges": {
    "ent_id":  "01.01.01.01",
    "burnr":   "0000000001",
    "fall_id": "F-2026-00001"
  },
  "result_data": "<spiges>…</spiges>"
}
```

Signature, Retry et DLQ du Webhook

Signé HMAC-SHA256, retries avec backoff, DLQ pour rejouer

Signature HMAC-SHA256

- En-tête : Authorization: HMAC-SHA256 t=<unix>,v1=<hex-sig>
- Signature = HMAC-SHA256(secret, "<t>.<corps-brut>")
- Émise uniquement si IntegrationProfile.webhookSigning=HMAC_SHA256
- Comparaison à temps constant ; rejeter si écart de t > 5 min (replay)

Plan de retry

- Par défaut : 4 tentatives à 30s, 2 min, 10 min, 1 h
- Retentables : 5xx, 408, 429, erreurs réseau
- Définitifs : 2xx (livré), 4xx (sauf 408/429)
- Configurable par profil : webhookRetryDelaysSeconds, webhookRetryMax

Dead-Letter Queue

- Sur échec final : ligne persistée dans WebhookFailure
- Lister : GET /api/v1/admin/webhook-failures
- Détail : GET /api/v1/admin/webhook-failures/{id}
- Rejouer : POST /api/v1/admin/webhook-failures/{id}/replay

```
# Replay a failed delivery
curl -X POST \
  -H "Authorization: Bearer <admin-jwt>" \
  https://kodemed/api/v1/admin/\
    webhook-failures/<id>/replay
```

KodeMed

Contact & support

info@kodemed.ch
www.kodemed.ch