KodeMed

Guida all’integrazione

Per sviluppatori software e integratori di sistema

Version 2026.5.15.59502 | KodeMed AG

Panoramica dell’integrazione

Due percorsi di integrazione verso KodeMed

Opzione A: DLL COM

Integrazione diretta nel SIO

- Caricare la DLL tramite variabile KODEMED_DLL_PATH
- Chiamare SendConfig() con XML o JSON
- Chiamare DoCodingWithFormat() per il flusso UI completo
- Chiamare DoCoding() per l’elaborazione headless
- Recuperare i risultati tramite GetResults()
- Funziona con .NET, VB6, VBA, C#, Delphi

Opzione B: API REST / WebSocket

Applicazioni di terze parti via CodingClient

- POST /api/v1/coding/session con i dati
- Il server invia al CodingClient via WebSocket
- CodingClient apre l’UI di codifica sul desktop dell’utente
- Interrogare lo stato della sessione via endpoint GET
- Qualsiasi linguaggio/piattaforma (Java, Python, ...)
- Aggiornamenti in tempo reale via WebSocket

Opzione A: Integrazione DLL COM

Passo dopo passo per applicazioni .NET / VB / C#

Esempio 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();
```

Esempio 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
```

Risoluzione percorso DLL

- Variabile KODEMED_DLL_PATH (impostata da CodingClient)
- COM: CreateObject("KodeMed.Coding")
- Reflection: Assembly.LoadFile(dllPath)

Opzione B: Integrazione API REST

Per qualsiasi linguaggio/piattaforma

Flusso degli endpoint

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

Esempio: Creare una sessione (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"}
```

Opzione C: Integrazione WebSocket

Comunicazione bidirezionale in tempo reale

Tipi di messaggi 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

Connessione

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

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

// Auto-reconnect: 60s
```

Formato payload (JSON)

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

Autenticazione

OAuth2 / OpenID Connect

Flusso OAuth2 / OIDC

- Compatibile con: Keycloak, Auth0, Azure AD, Okta
- Flusso Authorization Code con PKCE
- La DLL apre un browser integrato per il login
- Aggiornamento automatico dei token (access + refresh)
- Lingua utente letta dal claim JWT ‘locale’
- Client ID: kodemed-dll (DLL), kodemed-ui (browser)
- Realm: configurabile (predefinito: kodemed)

Configurazione

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

Ciclo di vita dei token

- Access token: breve durata (~5 min)
- Refresh token: lunga durata (~30 min)
- Aggiornamento automatico prima della scadenza
- Riconnessione WebSocket aggiorna il token

Formati dati

SpiGes XML, UST, Personalizzato

Struttura XML SpiGes (primaria)

```
<?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

Sistema plugin

- Implementare l’interfaccia IFormatPlugin
- Scoperta automatica dalla directory plugin
- Trasformare dati personalizzati nel modello SpiGes
- Integrato: JSON Clinical Plugin

Setup CodingClient

Applicazione system tray portatile

Installazione (3 passaggi)

- 1. Copiare la cartella KodeMed in qualsiasi posizione
- 2. Avviare KodeMed.CodingClient.exe
- 3. Auto-setup: variabili d’ambiente + registrazione DLL COM

Variabili d’ambiente (impostate automaticamente)

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

Compatibilità Citrix / VDI

- EXE autonomo: nessun runtime .NET richiesto
- Solo livello utente: tutte le scritture in HKCU
- Nessun diritto di amministratore richiesto
- Registrazione COM via HKCU\Software\Classes
- Config: accanto all’EXE o %KODEMED_HOME%
- VDI non persistente: re-setup ad ogni accesso

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

- Accesso / Disconnessione (OAuth2)
- Aprire sessione da file (SpiGes / UST / Personalizzato)
- Lingua (DE/FR/IT/EN)
- Impostazioni server, Avvio auto, Setup / Riparazione

Riferimento API rapido

Metodi DLL, Endpoint REST, Messaggi WebSocket

Metodi 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

Endpoint 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

Messaggi 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 lato DLL

Inviato da CodingClient — il vecchio flusso in-process

Come funziona

- Il codificatore completa la sessione → CodingClient invia JSON in POST
- Fire-and-forget: non blocca il codificatore
- Retry con backoff esponenziale (1s→2s→4s... max 60s)
- Nessun retry su 4xx (eccetto 429 Too Many Requests)

Configurazione server (variabili d'ambiente)

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

Configurazione client (auth locale)

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

Minimizzazione dati (GDPR/nLPD)

- includeResultData predefinito false (opt-in)
- includeGrouperResults predefinito false (opt-in)
- Credenziali auth mai inviate dal server
- Protezione SSRF sugli URL webhook

Contratto Webhook HIS REST

Invio lato server su case.coded / case.discarded

Trigger

- Il codificatore preme Apply → event_type = case.coded
- Il codificatore preme Discard → event_type = case.discarded
- Per-tenant: IntegrationProfile.webhookUrl + webhookEventTypes
- Emesso dal Server (NON dalla DLL) — per partner senza DLL

Idempotenza

- Header Idempotency-Key: <session_id>:<event_type>
- Stabile tra i retry — deduplica lato SIO tramite questo header
- Consegna at-least-once: il ricevente deve gestire i duplicati

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>"
}
```

Firma, Retry e DLQ del Webhook

Firmato HMAC-SHA256, retry con backoff, DLQ per replay

Firma HMAC-SHA256

- Header: Authorization: HMAC-SHA256 t=<unix>,v1=<hex-sig>
- Firma = HMAC-SHA256(secret, "<t>.<corpo-grezzo>")
- Inviato solo se IntegrationProfile.webhookSigning=HMAC_SHA256
- Confronto in tempo costante; rifiuta se t deriva > 5 min (replay)

Schedulazione Retry

- Predefinito: 4 tentativi a 30s, 2min, 10min, 1h
- Ripetibili: 5xx, 408, 429, errori di rete
- Permanenti: 2xx (consegnato), 4xx (eccetto 408/429)
- Configurabile per profilo: webhookRetryDelaysSeconds, webhookRetryMax

Dead-Letter Queue

- All'errore finale: riga persistita nella tabella WebhookFailure
- Elenco: GET /api/v1/admin/webhook-failures
- Dettaglio: GET /api/v1/admin/webhook-failures/{id}
- Replay: 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

Contatto & supporto

info@kodemed.ch
www.kodemed.ch