KodeMed

Integrationsleitfaden

Für Software-Entwickler & Systemintegratoren

Version 2026.5.15.59502 | KodeMed AG

Integrationsübersicht

Zwei Integrationswege zu KodeMed

Option A: COM-DLL

Direkte KIS-Integration

- DLL laden via KODEMED_DLL_PATH-Umgebungsvariable
- SendConfig() mit XML oder JSON aufrufen
- DoCodingWithFormat() für vollen UI-Ablauf aufrufen
- DoCoding() für Headless-Verarbeitung aufrufen
- Ergebnisse via GetResults() abrufen
- Funktioniert mit .NET, VB6, VBA, C#, Delphi

Option B: REST-/WebSocket-API

Drittanbieter-Apps via CodingClient

- POST /api/v1/coding/session mit Daten
- Server sendet an CodingClient via WebSocket
- CodingClient öffnet Kodierungs-UI auf dem Desktop
- Sitzungsstatus via GET-Endpunkt abfragen
- Jede Sprache/Plattform (Java, Python, ...)
- Echtzeit-Updates via WebSocket

Option A: COM-DLL-Integration

Schritt für Schritt für .NET-/VB-/C#-Anwendungen

C#-Beispiel

```
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();
```

VB.NET-/VBA-Beispiel

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

DLL-Pfadauflösung

- KODEMED_DLL_PATH-Umgebungsvariable (von CodingClient gesetzt)
- COM: CreateObject("KodeMed.Coding")
- Reflection: Assembly.LoadFile(dllPath)

Option B: REST-API-Integration

Für jede Sprache/Plattform

Endpunkt-Ablauf

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

Beispiel: Sitzung erstellen (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: WebSocket-Integration

Bidirektionale Echtzeit-Kommunikation

WebSocket-Nachrichtentypen

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

Verbindung

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

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

// Auto-reconnect: 60s
```

Payload-Format (JSON)

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

Authentifizierung

OAuth2 / OpenID Connect

OAuth2-/OIDC-Ablauf

- Kompatibel mit: Keycloak, Auth0, Azure AD, Okta
- Authorization-Code-Verfahren mit PKCE
- DLL öffnet eingebetteten Browser zur Anmeldung
- Token-Auto-Refresh (Access- + Refresh-Tokens)
- Benutzersprache aus JWT-„locale“-Claim
- Client-IDs: kodemed-dll (DLL), kodemed-ui (Browser)
- Realm: konfigurierbar (Standard: kodemed)

Konfiguration

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

Token-Lebenszyklus

- Access-Token: kurzlebig (~5 Min.)
- Refresh-Token: langlebig (~30 Min.)
- Auto-Refresh vor Ablauf
- WebSocket-Reconnect erneuert Token

Datenformate

SpiGes XML, BFS, Benutzerdefiniert

SpiGes-XML-Struktur (Primär)

```
<?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>
```

FormatType-Enum

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

Plugin-System

- IFormatPlugin-Interface implementieren
- Automatische Erkennung aus Plugin-Verzeichnis
- Benutzerdefinierte Daten in SpiGes-Modell umwandeln
- Integriert: JSON Clinical Plugin

CodingClient-Setup

Portable System-Tray-Anwendung

Installation (3 Schritte)

- 1. KodeMed-Ordner an beliebigen Ort kopieren
- 2. KodeMed.CodingClient.exe starten
- 3. Auto-Setup: Umgebungsvariablen + COM-DLL-Registrierung

Umgebungsvariablen (automatisch gesetzt)

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

Citrix-/VDI-Kompatibilität

- Eigenständige EXE: keine .NET-Runtime erforderlich
- Nur Benutzerebene: alle Schreibvorgänge in HKCU
- Keine Administratorrechte erforderlich
- COM-Registrierung via HKCU\Software\Classes
- Konfiguration: neben EXE oder %KODEMED_HOME%
- Nicht-persistentes VDI: Re-Setup bei jeder Anmeldung

Tray-Menü (mehrsprachig: DE/FR/IT/EN)

- Anmelden / Abmelden (OAuth2)
- Sitzung aus Datei öffnen (SpiGes / BFS / Benutzerdefiniert)
- Sprache (DE/FR/IT/EN)
- Servereinstellungen, Autostart, Setup / Reparatur

API-Kurzreferenz

DLL-Methoden, REST-Endpunkte, WebSocket-Nachrichten

DLL-Methoden (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

REST-Endpunkte

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

WebSocket-Nachrichten (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

Post-Coding-Webhook

Automatische Ergebnisübermittlung an Ihr KIS

Funktionsweise

- Kodierer schliesst Sitzung ab → CodingClient sendet JSON per POST
- Fire-and-Forget: blockiert den Kodierer nicht
- Exponentieller Backoff-Retry (1s→2s→4s... max 60s)
- Kein Retry bei 4xx (ausser 429 Too Many Requests)

Serverkonfiguration (Umgebungsvariablen)

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

Client-Konfiguration (lokale Auth)

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

Datenminimierung (DSGVO/nDSG)

- includeResultData standardmässig false (Opt-in)
- includeGrouperResults standardmässig false (Opt-in)
- Auth-Credentials werden nie vom Server gesendet
- SSRF-Schutz bei Webhook-URLs

HIS-REST-Webhook-Vertrag

Server-seitiger Versand bei case.coded / case.discarded

Auslöser

- Kodierer klickt Apply → event_type = case.coded
- Kodierer klickt Discard → event_type = case.discarded
- Pro Mandant: IntegrationProfile.webhookUrl + webhookEventTypes
- Wird vom Server gefeuert (NICHT von der DLL) — Partner ohne DLL nutzen diesen Pfad

Idempotenz

- Idempotency-Key Header: <session_id>:<event_type>
- Bleibt über Retries hinweg gleich — Deduplizierung HIS-seitig darüber
- At-least-once-Zustellung: Empfänger muss Duplikate verkraften

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>"
}
```

Webhook-Signatur, Retry & DLQ

HMAC-SHA256 signiert, Backoff-Retry, DLQ für Admin-Replay

HMAC-SHA256-Signatur

- Header: Authorization: HMAC-SHA256 t=<unix>,v1=<hex-sig>
- Signatur = HMAC-SHA256(secret, "<t>.<raw-body>")
- Nur gesendet, wenn IntegrationProfile.webhookSigning=HMAC_SHA256
- Constant-time-Vergleich; t-Skew > 5 min ablehnen (Replay-Schutz)

Retry-Plan

- Standard: 4 Versuche bei 30s, 2min, 10min, 1h
- Wiederholbar: 5xx, 408, 429, Netzwerkfehler
- Endgültig: 2xx (zugestellt), 4xx (außer 408/429)
- Pro Profil konfigurierbar: webhookRetryDelaysSeconds, webhookRetryMax

Dead-Letter-Queue

- Bei finalem Fehler: Zeile in WebhookFailure-Tabelle
- Liste: GET /api/v1/admin/webhook-failures
- Detail: 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

Kontakt & Support

info@kodemed.ch
www.kodemed.ch