INTEGRATION GUIDE · FOR HIS / PARTNER DEVELOPERS

KodeMed COM DLL Integration

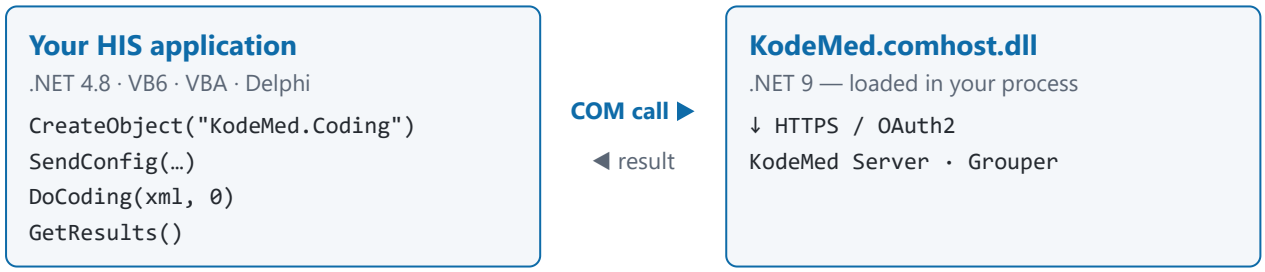
Call the KodeMed clinical-coding engine from an existing Windows application — C#, VB.NET, classic VB6/VBA, or Delphi — over standard COM. No rewrite of your HIS, no runtime upgrade.

Delivery: the KodeMed MSI in COM-only mode (`INSTALLCODINGCLIENT=0`) · SPIGES schema 1.4 / 1.5 · Document version 1.4

1. What this is, in one minute

KodeMed does the medical **coding** and **grouping** of a hospital case — *grouping* = assigning the case to its billing tariff group (Swiss DRG, TARPSY for psychiatry, ST Reha for rehabilitation). Your application hands KodeMed one case as **SPIGES XML** (the Swiss BFS hospital-case format — Section 9); a coder reviews and completes it (or it runs **headless** — automatically, with no coder screen); then **your application reads the coded result back and stores it in your HIS**. That read-back is the whole point of the integration (see Section 5).

The engine ships as a Windows DLL that registers as a normal **COM server**. Your program creates it by name (`KodeMed.Coding`), calls four methods, and reads the result. If your code can already do `CreateObject("Excel.Application")` , it can call KodeMed the same way.



Why COM? The engine is built on .NET 9. Your application does not need .NET 9 and does not reference the engine directly. COM loads the .NET 9 runtime *inside your process, side by side* with whatever runtime you already use. You keep your .NET 4.8 / VB6 / Delphi codebase untouched.

2. Before you start — prerequisites

You need	Detail
Windows 10 or 11 (or Server 2019+)	Desktop Windows with a UI session — the coder login uses an embedded browser.
.NET Desktop Runtime 9 (x64)	The **Desktop** runtime, x64 — **not** the plain ".NET Runtime" and not the SDK. Direct installer: `aka.ms/dotnet/9.0/windowsdesktop-runtime-win-x64.exe`. Without it, creating the object fails with `0x80008096`. Verify with `dotnet --list-runtimes` → a `Microsoft.WindowsDesktop.App 9.x` line must appear.
Administrator rights — once	Only to install the MSI machine-wide (Section 3). A per-user install needs no admin. Running the app afterwards needs no admin.
A KodeMed account	Coders sign in with their KodeMed / hospital login (OAuth2). Your integration does not handle passwords.

> **Bitness — important.** The COM server is **x64 only**, and a COM DLL loads *in-process*, so **your calling application must be a 64-bit process**. Modern .NET apps and 64-bit Office/VBA are fine. Calling from a 32-bit process gives `0x80040154` "class not registered" even after a successful install. A **32-bit-only host — notably classic VB6** — cannot load the x64 in-process server; for those, integrate server-to-server via the **HIS REST** API instead (the *HIS REST Integration Spec*, §10) rather than COM. The VB6/VBA snippets below show the COM call shape and apply to **64-bit** VBA/hosts.

3. Install the COM DLL (once per machine)

The COM component ships inside the **KodeMed MSI**. Installing it in **COM-only mode** drops the DLL and **registers the COM object for you** — no extracting, no `regsvr32`, no `Unblock-File`.

1. **Install the prerequisite**
   — the .NET Desktop Runtime 9 (x64) from the link in Section 2. (The MSI installs fine without it, but COM only works once it is present.)

2. **Download the MSI**
   (public, no login):

   ```
   https://demo-portal.kodemed.ch/windows/KodeMed.msi
   ```

3. **Install in COM-only mode**
   (no coding-client app). From a command prompt:

```
REM Machine-wide (recommended for a HIS server) — elevated prompt, silent:
msiexec /i KodeMed.msi INSTALLCODINGCLIENT=0 ALLUSERS=1 /qn

REM Per-user (no admin; registers under HKCU):
msiexec /i KodeMed.msi INSTALLCODINGCLIENT=0 /qn
```

The MSI registers COM per-user (HKCU) or machine-wide (HKLM) to match the install scope — so there are no HKCU/HKLM conflicts. Uninstall via *Settings › Apps* removes the registration cleanly. Fleet rollout: deploy the MSI per-machine via Group Policy with `INSTALLCODINGCLIENT=0 ALLUSERS=1`.

The install publishes these identifiers. You normally only need the **ProgId**.

Identifier	Value	Used for
ProgId	`KodeMed.Coding`	`CreateObject(...)` — the easy way
CLSID (class)	`{F6E73053-8D49-480F-AC06-4E0A4882373F}`	typed / early-bound clients
Interface `IKodeMed`	`{B0FDE709-7A9D-4F22-BD24-BFE56BE80E87}`	typed clients
Events `IKodeMedEvents`	`{F30E7A47-F155-4F24-A555-9A6448DC8C71}`	optional progress/error callbacks

4. The integration — four calls

Whatever the language, the flow is always the same:

1. **Create the object**
   — `CreateObject("KodeMed.Coding")`.

2. **Configure it once**
   — `SendConfig(configText)`: the URL of your KodeMed server (Section 7). Returns `True` on success.

3. **Process one case**
   — pass the SPIGES XML:
   `DoCoding(xml, 0)` — headless: sends the case straight to the grouper, no coder screen.
   Note:
   the very first sign-in still opens a browser window once for the OAuth login; after that it is silent.
   `DoCodingWithFormat(xml, 0)` — opens the coding screen for a human, waits until they finish.
   (`0` = SPIGES format. Both return `True` on success.)

4. **Read the result back and store it**
   — `GetResultData()` for the coded case XML, or `GetResults()` for the full envelope (coded case + DRG). This read-back is the point of the whole exercise; Section 5 shows exactly where the data is and when to save it. Also useful: `LastError`, `LastCodingAction`.

Enums cross COM as plain integers. **FormatType:** `0` =SPIGES, `1` =BFS, `2` =Custom. **CodingAction:** `0` =Pending, `1` =Applied, `2` =Discarded, `3` =Cancelled, `4` =Timeout, `5` =Error. Call `DisposeClient()` when you are done with the object.

Sign-in and sign-out: the coder signs in once; the DLL keeps that sign-in for later cases, and a `SendConfig()` with the same server keeps it too. When the user signs out of your HIS, call `Logout()`: it closes the coder's KodeMed sign-in (identity-provider session and tokens), so the next `DoCoding` / `DoCodingWithFormat` asks for a login again. Use it whenever another person may sit down at the HIS next. `Logout()` exists from `KodeMed.dll` file version 2026.10.22.2579 (release 2026.10.22.55417); an older `KodeMed.dll` answers "member not found", so wrap the call in your error handling if your users may still run an older client.

Session timeout: a coding session ends after a period without coder activity (60 minutes by default, set on the server); every action of the coder pushes that limit forward. The coder sees a warning 5 minutes before. When a session ends this way the coding window closes and `LastCodingAction` is `4` (Timeout): the case is unchanged, **save nothing**, and open it again if the coder is still working on it. A `KodeMed.dll` with a file version below 2026.10.22.2583 (release 2026.10.22.78706) reports `3` (Cancelled) here, including 2026.10.22.2579 of the same day; handle `3` and `4` the same way. The file version is in the properties of `KodeMed.dll` (Details tab).

Threading: because the sign-in uses an embedded browser, run KodeMed on a UI (STA) thread. In C# mark your entry point `[STAThread]` (as the sample does). The DLL falls back to its own STA thread if you don't, so a service/MTA caller still works — but STA is the tested path.

5. What the DLL returns — read it and store it

The DLL does not write anything back into your HIS for you — you read the result out and save it. And there are **two distinct results** (integrations often confuse them):

- **1 · The coded / modified SPIGES case** — the same case you sent, now with the coding applied (added / edited diagnoses & procedures). You write this back **over your original case**. It is SPIGES XML.
- **2 · The grouping result** — what the grouper computed *from* that case: the SwissDRG / TARPSY / ST Reha code, its description, MDC, cost weight, etc. This is **separate metadata** you store alongside the case (for billing) — it is **not** part of the SPIGES.

Two calls give them to you — pick by what you need:

The simple way — you just want the coded case back

Call `GetResultData()`. It returns the **coded SPIGES XML of your case, directly** — no envelope to parse, always XML. Store that string in your HIS:

```
if (km.DoCoding(spigesXml, 0)) {
    string codedSpiges = km.GetResultData();   // <- the coded SPIGES XML: SAVE THIS
}
```

(For a single `<Fall>` — the normal case — this **is** your coded case. VB6/VBA/Delphi: `coded = km.GetResultData()`.)

The full way — you also want the DRG / grouping result

Call `GetResults()`. It returns a **results envelope** (XML by default, JSON if you set `OutputFormat`) with, per case, both the coded SPIGES **and** the grouper outcome:

```
<codingResults xmlns="https://schemas.kodemed.ch/dll_com/GetResults-1.0"
               occurredAt="…" validationMode="warn">
  <cases>
    <case format="spiges">                          <!-- caseId/siteId are on codingResult
      <data><!-- the CODED SPIGES XML for this Fall (XML-escaped) --></data>
      <codingResult caseId="1" siteId="10000001" tariff="SWISSDRG" drg="…" mdc="…" partitio
        <pccl>…</pccl><effectiveCostWeight>…</effectiveCostWeight><los>…</los>
        <!-- + diagnoses / procedures (used/valid/excludedFromGrouping), stats -->
        <splg splg="…">…</splg>   <!-- SPLG/GPPH (Kanton ZH) — present when the SPLG groupe
      </codingResult>
    </case>
  </cases>
</codingResults>
```

Read it like any XML — one `<case>` per `<Fall>` you sent; `<data>` = the coded SPIGES to store, `<codingResult>` = the DRG / tariff for billing:

```
// C#
var doc = System.Xml.Linq.XDocument.Parse(km.GetResults());
System.Xml.Linq.XNamespace ns = "https://schemas.kodemed.ch/dll_com/GetResults-1.0";
foreach (var c in doc.Descendants(ns + "case")) {
    string codedSpiges = c.Element(ns + "data")?.Value;              // SAVE THIS
    string drg         = c.Element(ns + "codingResult")?.Attribute("drg")?.Value;
}
```

Raw example — a headless grouping call (no UI)

Send the sample case from Section 9 to `DoCoding(xml, 0)` (no coder screen), then `GetResults()` returns an envelope like this — the two results side by side: the coded SPIGES in `<data>` (result 1) and the grouper's output in `<codingResult>` (result 2):

```
<codingResults xmlns="https://schemas.kodemed.ch/dll_com/GetResults-1.0"
               occurredAt="2026-09-18T18:41:00+02:00" validationMode="warn">
  <cases>
    <case format="spiges">                        <!-- caseId/siteId are on codingResult --

      <!-- RESULT 1: the coded SPIGES for Fall 1 (the DLL re-serialises it; XML-escaped) --
      <data>
        &lt;Unternehmen xmlns="http://www.bfs.admin.ch/xmlns/gvs/spiges-data" ent_id="10000
          &lt;Standort burnr="10000001"&gt;&lt;Fall fall_id="1"&gt;
            &lt;Administratives abc_fall="A" alter="38" … eintritt_aufenthalt="1"
                           eintrittsdatum="2026061010" austrittsdatum="2026061209" …/&gt;
            &lt;Diagnose diagnose_id="1" diagnose_kode="K3530" diagnose_poa="3"/&gt;
            &lt;Diagnose diagnose_id="2" diagnose_kode="C181" diagnose_poa="3"/&gt;
            &lt;Behandlung behandlung_id="1" behandlung_chop="4701"/&gt;
            &lt;Behandlung behandlung_id="2" behandlung_chop="4573"/&gt;
          &lt;/Fall&gt;&lt;/Standort&gt;&lt;/Unternehmen&gt;
      </data>

      <!-- RESULT 2: the grouper output (SwissDRG 15.0) -->
      <codingResult caseId="1" siteId="10000001" tariff="SWISSDRG" drg="G22D" mdc="06"
                    partition="O" grouperVersion="SwissDRG 15.0" grouperStatus="NORMAL_GROU
        <pccl>2</pccl>
        <costWeight>0.699</costWeight>
        <effectiveCostWeight>0.699</effectiveCostWeight>
        <los>2</los>  <ltp>1</ltp>  <htp>4</htp>  <groupingSuccess>true</groupingSuccess>
        <diagnoses>
          <diagnosis id="1" code="K3530" catalog="ICD-10-GM">
            <isHauptdiagnose>true</isHauptdiagnose><used>true</used><valid>true</valid></di
          <diagnosis id="2" code="C181" catalog="ICD-10-GM">
            <ccl>3</ccl><used>false</used><valid>true</valid><excludedFromGrouping>true</ex
        </diagnoses>
        <procedures>
          <procedure id="1" code="4701" catalog="CHOP"><used>true</used><valid>true</valid>
          <procedure id="2" code="4573" catalog="CHOP"><used>false</used><valid>false</vali
        </procedures>
        <!-- SPLG / GPPH (GDZH Spitalplanungs-Leistungsgruppen, Kanton Zürich) — additive;
             when the SPLG grouper is licensed/active and assigns a group. Absent otherwise
        <splg burnr="10000001" fallId="1" splg="BP">
          <allSplg>BP</allSplg>
          <lactrl>99</lactrl>
          <slst>0</slst>
        </splg>
      </codingResult>
    </case>
```

```
  </cases>
  <stats casesProcessed="1" diagnosesCount="2" treatmentsCount="2" spiGesVersion="1.5" succ
    <enterpriseId>100000001</enterpriseId></stats>
</codingResults>
```

This is a real capture — a headless `DoCoding` of the sample case against the KodeMed demo (SwissDRG 15.0): DRG **G22D**, MDC 06, partition O, PCCL 2, cost weight **0.699**, LOS 2. In this run the sample had pre-excluded `diagnose_id=2` / `behandlung_id=2` via `SetGroupingExclusions` (Section 8) — hence `used=false` + `excludedFromGrouping=true` on those. The DLL serialises the grouping numbers as **child elements** and identifiers/DRG as attributes; the `<data>` body is the SPIGES **re-serialised and XML-escaped** (un-escape it, or just call `GetResultData()`, to get result 1). A headless call carries no `eventType` and `LastCodingAction=Pending` (no coder) — see below.

SPLG / GPPH. When the case is grouped for the GDZH Spitalplanungs-Leistungsgruppen (Kanton Zürich) and the SPLG grouper is licensed and active, each `<codingResult>` also carries a `<splg>` block (as above). It is **additive**: absent when the SPLG grouper is on standby or assigns no group, and it never blocks or alters the DRG result. Fields — attributes `splg` (the primary Leistungsgruppe, e.g. `BP`, `BEW1`), `burnr`, `fallId`, `notes`, `zusatz`; child elements `<allSplg>` (all matched groups, repeatable), `<quer>` / `<mfzs>` / `<mfzo>` (Reha/Psychiatrie groups), `<errors>` (vendor codes), `<lactrl>` (Leistungscontrolling status — e.g. `99` when the site's BUR is not in the loaded Spitalliste), and `<slst>` (Status Spitalliste, the "Stsl" flag: vendor field `slst`, passed through as returned by the SPLG grouper, #2119). Read it just like the DRG: `c.Element(ns + "codingResult")?.Element(ns + "splg")?.Attribute("splg")?.Value`.

Same case, interactive (`DoCodingWithFormat`) — the coder modified it

Run the same case through `DoCodingWithFormat`: the coder opened it, **added** a diagnosis (`T8902`) and a procedure (`920910`) and clicked Apply. Now the envelope carries `sessionId` + `eventType="case.coded"`, `LastCodingAction=Applied`, and — the key point — **the `<data>` is the coder's modified case** (three diagnoses, three procedures):

```
<codingResults xmlns="https://schemas.kodemed.ch/dll_com/GetResults-1.0"
               sessionId="7d682abf-9ae6-4bd0-8153-44c91fe7f058" eventType="case.coded"
               occurredAt="2026-09-18T18:56:03+02:00" validationMode="warn">
  <cases>
    <case id="1" siteId="10000001" format="spiges">
      <data>
        &lt;Unternehmen ent_id="100000001" version="1.5" xmlns="…/spiges-data"&gt;&lt;Stand
          &lt;Administratives abc_fall="A" alter="38" … /&gt;
          &lt;Diagnose diagnose_id="1" diagnose_kode="K3530" diagnose_poa="3" /&gt;
          &lt;Diagnose diagnose_id="2" diagnose_kode="C181" diagnose_poa="3" /&gt;
          &lt;Diagnose diagnose_id="3" diagnose_kode="T8902" /&gt;                &lt;!--
          &lt;Behandlung behandlung_id="1" behandlung_chop="4701" behandlung_beginn="202606
          &lt;Behandlung behandlung_id="2" behandlung_chop="4573" behandlung_beginn="202606
          &lt;Behandlung behandlung_id="3" behandlung_chop="920910" /&gt;           &lt;!-
        &lt;/Fall&gt;&lt;/Standort&gt;&lt;/Unternehmen&gt;
      </data>
      <codingResult caseId="1" siteId="10000001" tariff="SWISSDRG" drg="G22D" mdc="06"
                    partition="O" grouperVersion="SwissDRG 15.0" grouperStatus="NORMAL_GROU
        <pccl>0</pccl>  <costWeight>0.699</costWeight>  <effectiveCostWeight>0.699</effecti
        <!-- diagnoses / procedures now list all three each (with used/valid/excludedFromGr
      </codingResult>
    </case>
  </cases>
</codingResults>
```

Real capture from the demo. The added `T8902` / `920910` did not change the DRG here (still G22D), but they **are** in the returned `<data>` — so because `LastCodingAction=Applied` you store this modified case over your original. (Note the interactive envelope also carries `sessionId` and, on the `<case>`, `id` / `siteId`, which a headless call omits.)

When to save — it depends on which call you used. Check `LastCodingAction` before writing anything:

Headless `DoCoding`: there is no coder, so `LastCodingAction=Pending` and the envelope has no `eventType`. The grouping ran — **use the result** (store the coded `<data>` + the `<codingResult>`).

Interactive `DoCodingWithFormat`: `LastCodingAction` tells you what the coder did — `Applied` (`eventType="case.coded"`) → **store the coded result** (the `<data>` now reflects the coder's changes); `Discarded` / `Cancelled` / `Timeout` (`session.expired`) → the coder kept the original, **save nothing**.

Validation: on a demo/trial licence the grouper runs in *warn* mode — the case is still coded even if the SPIGES isn't 100% schema-clean, and any issues are listed in a `<validationWarnings>` block in the envelope. Fix those before production.

6. Copy-paste examples

Fill two strings first. Every snippet below uses `configXml` (the configuration text from **Section 7**) and `spigesXml` (one case in the format from **Section 9**). Declare each as a `String` and load it — for a first test, paste the ready-made samples from those sections. **Even faster:** download the `KodeMed.Com.Test` sample (Section 10) and run `run-sample.cmd` — it does all of this against a demo server, then adapt the snippet for your language.

Use late binding — there is no type library. The component is accessed by name through `IDispatch` (`CreateObject` / `CreateOleObject` / C# `dynamic`), exactly as the snippets below do. It intentionally ships **without a** `.tlb`, so **early binding fails** — Visual Studio's *Add Reference → COM*, or PowerShell `New-Object -ComObject "KodeMed.Coding"`, error with `0x80131165` / "type library not found". That is expected; you do not need a typelib — just use late binding.

Classic VB6 / VBA (late-bound — the simplest)

```
Dim km As Object
Dim configXml As String, spigesXml As String, result As String
' configXml = ...   (from Section 7)   spigesXml = ...   (from Section 9)
Set km = CreateObject("KodeMed.Coding")

If Not km.SendConfig(configXml) Then
    MsgBox "Config failed: " & km.LastError
    Exit Sub
End If

' 0 = SPIGES, headless. (For the coder screen use DoCodingWithFormat - note below.)
If km.DoCoding(spigesXml, 0) Then
    result = km.GetResultData()       ' the CODED SPIGES XML -> SAVE THIS
Else
    MsgBox "Coding failed: " & km.LastError
End If
' Interactive: save ONLY when the coder applied (LastCodingAction = 1 = Applied):
'   If km.DoCodingWithFormat(spigesXml, 0) And km.LastCodingAction = 1 Then result = km.Get

km.DisposeClient
Set km = Nothing
```

Delphi (late-bound via IDispatch)

```
uses ComObj, Variants;
var
  KM: Variant;
  Coded: string;
begin
  KM := CreateOleObject('KodeMed.Coding');
  if not KM.SendConfig(ConfigXml) then
    raise Exception.Create('Config failed: ' + KM.LastError);

  if KM.DoCoding(SpigesXml, 0) then     // 0 = SPIGES, headless
    Coded := KM.GetResultData          // the CODED SPIGES XML - store it
  else
    raise Exception.Create('Coding failed: ' + KM.LastError);
  // Interactive: if KM.DoCodingWithFormat(SpigesXml,0) and (KM.LastCodingAction = 1) then

  KM.DisposeClient;
end;
```

C# / VB.NET

Easiest is late-bound, like above (needs a reference to `Microsoft.CSharp` for `dynamic` on older .NET 4.8 projects):

```
Type t = Type.GetTypeFromProgID("KodeMed.Coding");
dynamic km = Activator.CreateInstance(t);

string configXml = /* the config text, Section 7 */;
string spigesXml = /* one case, Section 9 */;

if (!km.SendConfig(configXml)) throw new Exception(km.LastError);
if (!km.DoCoding(spigesXml, 0)) throw new Exception(km.LastError);   // 0 = SPIGES, headles

string codedCase = km.GetResultData();   // the CODED SPIGES XML — SAVE THIS
km.DisposeClient();

// Interactive coder screen — save ONLY when applied (1 = Applied):
// if (km.DoCodingWithFormat(spigesXml, 0) && (int)km.LastCodingAction == 1)
//     codedCase = km.GetResultData();
```

Want compile-time checked calls, events, or the per-case grouping exclusions? Use the ready-made, one-file typed wrapper from the official sample — copy `KodeMedCom.cs` into your project and call `new KodeMedClient()`. Full sample apps (both .NET 4.8-over-COM and .NET 9): see Section 10.

7. Configuration — you only need your server URL

From a developer's point of view the question is simply "*what do I have to configure?*" — and the answer is short: **the URL of your KodeMed server, and nothing else.** The DLL asks that server for everything else it needs (the login settings, the DataServer, the Grouper, the WebSocket, the coding UI). You never hand-copy those.

Step 1 — find your server URL, and confirm it in a browser

Your KodeMed **Server** is the REST/API endpoint:

Your setup	Server URL looks like
Hosted / SaaS	`https://server.kodemed.your-hospital.ch` — KodeMed AG gives you this.
On-prem, single host	`https://your-host:8443` — the **API port 8443**, **not** `:443` (that is the web UI).

Confirm it before you write any code: open `<your-server-url>/api/v1/config` in a web browser.

- **You get JSON** (a block of settings) → your server URL is correct, and your configuration is essentially done.
- **You get 404 / a login page / nothing** → that URL is wrong. Fix it first; nothing else will work until this returns JSON.

That JSON is exactly what the DLL reads — so `/api/v1/config` is both your test that the URL is right and the list of what the server will supply automatically.

Step 2 — give that URL to the DLL (pick one)

A) No code — a config file next to the DLL (simplest, recommended for a HIS). Put a file named `kodemed-client-config.json` in the **same folder as** `KodeMed.dll` ; the DLL loads it automatically at startup — you never call `SendConfig` :

```
{
  "serverUrl": "https://your-host:8443"
}
```

(A `kodemed-client-config.xml` with root `<KodeMedConfig>` works too.) A later `SendConfig()` from your code still overrides this file.

Where the DLL looks (first hit wins; `.json` before `.xml`): 1) the environment variable `KODEMED_CLIENT_CONFIG` : the full path of a config file (e.g. `H:\Kodemed\kodemed-client-config.json` on a user share) or of a folder holding one (`H:\Kodemed`); `%VARS%` are expanded; use an absolute path - a relative one is resolved against the host application's working directory, 2) the folder of `KodeMed.dll` , 3) the folder of the host application's EXE (older behaviour, kept for compatibility). The KodeMed log shows the file it loaded (`Auto-loading config from DLL folder: ...`) or every place it searched.

Developing against our docker-compose (or any self-signed / IP server)? The local KodeMed compose — and any on-prem "IP / self-signed" install — serves a **self-signed TLS certificate**. The DLL rejects that by default, so REST calls and the login window fail. Add one flag so it accepts it:

```
{
  "serverUrl": "https://localhost:8443",
  "trustSelfSignedCertificates": true
}
```

`trustSelfSignedCertificates` covers both the REST calls and the embedded login browser. **Testing / development only** — never enable it in production; there, use a real certificate (or import your server's CA into the Windows store). Use plain `http://` only for `http://localhost`.

On an **IP / self-signed** dev box (the Caddy preset), if opening `<serverUrl>/api/v1/config` shows the server does **not** publish the per-service URLs, list them yourself: `serverUrl` `:8443`, `codingUIUrl` `:443`, `dataServerUrl` `:8444`, `grouperServerUrl` `:8445`, `oauth2Url` `:9443`.

B) In code — `SendConfig(...)`. Pass XML (root `<KodeMedConfig>`) or JSON (root `{ }`); the DLL auto-detects by the first character. Minimal (add `<TrustSelfSignedCertificates>true</TrustSelfSignedCertificates>` for a self-signed dev server):

```
<KodeMedConfig><ServerUrl>https://your-host:8443</ServerUrl></KodeMedConfig>
```

What the server fills in for you (so you don't have to)

On first connect the DLL calls `GET <ServerUrl>/api/v1/config` and reads back the URL and login rows below — you never set those. List one in your own config **only** to override what the server publishes (rarely needed). The last two rows are **local client options**, not published by the server:

Field (XML element)	JSON property	Set it?	Meaning
`ServerUrl`	`serverUrl`	**always**	The one field you set — everything below is derived from it
`OAuth2Url` / `OAuth2Realm` / `OAuth2ClientId`	`oauth2Url` / `oauth2Realm` / `oauth2ClientId`	auto	Login (OAuth2) settings — the server publishes them
`WebSocketUrl`	`webSocketUrl`	auto	Live session channel
`CodingUIUrl`	`codingUIUrl`	auto	Coder screen — only used by `DoCodingWithFormat`
`DataServerUrl`	`dataServerUrl`	auto	Thesaurus / catalogues host
`GrouperServerUrl`	`grouperServerUrl`	auto	DRG / TARPSY / ST Reha host
`OutputFormat`	`outputFormat`	optional	`XML` (default) or `JSON` — controls `GetResults()`
`SpiGesVersion`	`spiGesVersion`	optional	`1.5` (default) or `1.4`

Only if you set the OAuth2 fields by hand (instead of letting the server publish them), two traps:

1. **OAuth2 fields are flat siblings** — `OAuth2Url` + `OAuth2Realm` + `OAuth2ClientId`. Do **not** nest them in an `<OAuth2>` block, or logins go without a token and the server returns 401.

2. `OAuth2Url` **is the login server base, not the realm URL.** A *realm* is the login server's tenant name (KodeMed uses `kodemed`). The DLL builds `{OAuth2Url}/realms/{realm}/.well-known/openid-configuration` itself. Paste it in a browser: JSON = right, 404 = wrong. (Legacy Keycloak ≤16 needs an extra `/auth` in the base.)

8. Optional: pre-exclude codes from grouping

Your HIS can tell KodeMed, per case, which codes must stay **out of grouping** before the coder opens the case. Call this **before** `DoCoding` / `DoCodingWithFormat`; the coder screen then shows those codes already excluded.

```
' VB6 / VBA — a normal integer array; the runtime does the rest
Dim diag(0) As Long, proc(0) As Long
diag(0) = 2 : proc(0) = 2
ok = km.SetGroupingExclusions("10000001", 1, diag, proc)
km.ClearGroupingExclusions            ' reset between unrelated cases
```

```
// Delphi
KM.SetGroupingExclusions('10000001', 1, VarArrayOf([2]), VarArrayOf([2]));
```

- `burnr` — the Standort BUR number (text, max 9).
- `fall_id` — the SPIGES case id (a whole number).
- the two arrays — the `diagnose_id` / `behandlung_id` values (0–9999) to leave out. An empty array means "exclude nothing" on that side.

C# note: the two array parameters marshal as `SAFEARRAY(VT_I4)`. In a hand-written typed C# interface declare them as `[MarshalAs(UnmanagedType.SafeArray, SafeArraySubType = VarEnum.VT_I4)] int[]` (already done for you in the sample's `KodeMedCom.cs`). VB6/VBA/Delphi need no such attribute — pass a normal array.

9. SPIGES — the data you send and receive

SPIGES ("Spitalstationäre Gesundheitsversorgung") is the Swiss Federal Statistical Office (BFS) data collection for inpatient hospital cases, mandatory since 1 January 2024. It is the **public, standard** format KodeMed reads and writes — you do not invent a KodeMed-specific format.

Public definition — where the format comes from

SPIGES is defined publicly by the BFS as an **XSD schema** (the version KodeMed targets is `do-d-14.04-SPIGES`, XML namespace `http://www.bfs.admin.ch/xmlns/gvs/spiges-data`, schema versions `1.4` and `1.5`). The specification, the *Detailkonzept* and the schema are published on the BFS website:

Source	Where
BFS — Projekt SpiGes (official page)	`bfs.admin.ch → Statistiken → Gesundheit → Gesundheitswesen → Projekt SpiGes`
BFS — Krankenhausstatistik / Detailkonzept ab SpiGes	the *Detailkonzept* publication (contains the record layout & XSD)
KodeMed — SPIGES Data-Model Reference	a partner-friendly table of every element & attribute, generated from the BFS XSD — ask KodeMed AG or see the downloads portal

Because SPIGES is a public standard, if your HIS already produces the BFS inpatient export, you very likely already have the XML KodeMed needs.

The shape KodeMed uses

```
Unternehmen            (the hospital operator — root element)
└─ Standort           (a site; carries burnr)
   └─ Fall            (one case; carries fall_id)
      ├─ Administratives   admission/discharge, age, sex, canton …
      ├─ Diagnose*         ICD-10 codes  (diagnose_id, diagnose_kode, diagnose_poa)
      └─ Behandlung*       CHOP procedures (behandlung_id, behandlung_chop, …)
```

(The full schema also allows `Medikament` [ATC], `Neugeborene`, `Psychiatrie` [TARPSY], billing and canton blocks — KodeMed reads them all; a minimal coding case needs only the elements above.)

A minimal valid case (this really runs)

```
<?xml version="1.0" encoding="UTF-8"?>
<Unternehmen xmlns="http://www.bfs.admin.ch/xmlns/gvs/spiges-data"
             ent_id="100000001" version="1.5">
  <Standort burnr="10000001">
    <Fall fall_id="1">
      <Administratives abc_fall="A" admin_urlaub="0" alter="38" geschlecht="1"
          burnr_gesv="10000001" eintritt_aufenthalt="1"
          eintrittsart="1" eintrittsdatum="2026061010"
          austrittsdatum="2026061209" austrittsentscheid="1"
          wohnkanton="AG" wohnland="CHE" nationalitaet="CHE"
          grundversicherung="1" liegeklasse="1"/>
      <Diagnose diagnose_id="1" diagnose_kode="K3530" diagnose_poa="3"/>
      <Diagnose diagnose_id="2" diagnose_kode="C181" diagnose_poa="3"/>
      <Behandlung behandlung_id="1" behandlung_beginn="202606110000" behandlung_chop="4701"
      <Behandlung behandlung_id="2" behandlung_beginn="202606110000" behandlung_chop="4573"
    </Fall>
  </Standort>
</Unternehmen>
```

You send this to `DoCoding` / `DoCodingWithFormat`; the coded result comes back through `GetResults()` (Section 5). Every attribute the schema *requires* for a coding case is present above (the rest are common optional fields, shown for realism). Dates are `YYYYMMDD` with an optional hour — `YYYYMMDD[HH]` for the Administratives dates, `YYYYMMDD[HHMM]` for `behandlung_beginn`.

10. Troubleshooting & getting the samples

Symptom	Cause & fix
`0x80040154` "class not registered" on `CreateObject`	The MSI isn't installed, or your process is 32-bit against the x64 COM server. Install the MSI COM-only (Section 3); run your app as x64.
`0x80008096` when creating the object	The **.NET Desktop Runtime 9 (x64)** is missing (the plain ".NET Runtime" is not enough). Install it — Section 2 — then it works, no reinstall.
`0x80008093`	The runtime config can't be read — usually a stale per-user (HKCU) COM registration shadowing the machine-wide MSI one, or a broken install. Re-run the MSI (Section 3); if it persists, delete the stale key `HKCU\Software\Classes\CLSID\{F6E73053-8D49-480F-AC06-4E0A4882373F}`. (`0x80008083` = no .NET at all → install the Desktop Runtime 9, Section 2.)
Server returns **401** after login	OAuth2 config wrong — check gotchas 1 & 2 in Section 7; open the `.well-known/openid-configuration` URL in a browser.
Login screen shows "Resource not found"	`OAuth2Url` missing `/auth` (legacy Keycloak) or a realm typo.
`BadImageFormatException` / `0x800700C1`	A 32-bit process tried to load the x64 COM server. Run your app as x64. A 32-bit-only host (classic VB6) cannot use the in-process COM server — integrate via the HIS REST API instead.
`0x80131165` / "type library not found" / `New-Object -ComObject` fails	You tried to use the component **early-bound**. It ships without a type library by design — use **late binding** (`CreateObject` / `CreateOleObject` / C# `dynamic`), as in Section 6. Do not add a COM reference.

> **A network or login error during coding still proves your integration works** — it means the DLL loaded in-process and answered back across COM. Fix the config, not the wiring.

Working sample projects (Apache 2.0 — copy freely)

Sample	What it is	Get it
`KodeMed.Com.Test`	.NET Framework 4.8 driving the DLL over COM. The whole contract is one file, `KodeMedCom.cs` — the canonical copy-paste for VB.NET / VBA / VB6 / Delphi too. Ships a one-click `run-sample.cmd` and a pre-built `.exe`.	Sources: demo-portal.kodemed.ch/samples/KodeMed.Com.Test-sources.zip Repo: www.mieresit.com/gitbucket/kodemed/KodeMed.Com.Test
`KodeMed.Test.UI`	.NET 9 reference app (direct reference, no COM) — the same flow, if you are already on .NET 9.	Sources: demo-portal.kodemed.ch/samples/KodeMed.Test.UI-sources.zip Repo: www.mieresit.com/gitbucket/kodemed/KodeMed.Test.UI

Both repos are public (Apache 2.0). The COM DLL bundle (MSI) and this guide are under demo-portal.kodemed.ch → the **COM DLL** section. The server-to-server REST option (no DLL) is documented separately in the *HIS REST Integration Spec*.

KodeMed AG — COM DLL Integration Guide. Identifiers, config fields and the SPIGES sample in this document are verified against the published KodeMed COM sample (`KodeMed.Com.Test`). SPIGES is a public standard of the Swiss Federal Statistical Office (BFS). Questions: `support@kodemed.ch`.