

Spickzettel (.NET)

👤 Rolle

Verwaltungssystem

Onlinedienst

📦 SDK · Modul

.NET · Fitko.FitConnect.Client

✓ Voraussetzungen

Keine – diese Seite ist zum Nachschlagen gedacht, nicht zum linearen Lesen. Für erklärte Schritt-für-Schritt-Rezepte siehe [Einrichten](#).

STAND

4.0.0-rc.1

.NET 10

Fitko.FitConnect.Client

API v3

TL;DR – Alle Kernbausteine des .NET-SDKs auf einer Seite: DI-Registrierung, Rollen-Clients, Senden, Empfangen, Antworten (BiDiKo), Cases & Eventlog, Directory, Verwaltung, Krypto, Validierung, Konfiguration, Fehlerbehandlung, Beispiele und die häufigsten Fallstricke.

📌 STAND DIESER SEITE

4.0.0-rc.1 des .NET-SDKs aus dem Repository [meta-sdk-dotnet](#): modulare NuGet-Pakete unter `Fitko.FitConnect.*`, DI-zentriert, mit denselben Rollen-Clients wie das Java-SDK (`AsOrganisation()` / `AsOnlineService()`). Der Release-Candidate-Status bedeutet: Signaturen können sich bis zum finalen Release noch ändern.

IOnlineServiceClient

`client.AsOnlineService()` · Typ C

- Sendet **Anträge** (`SendSubmission`) und **vorverschlüsselte** Anträge (`SendEncryptedSubmission`)
- Liest **Antworten**: `FetchAvailableReplies`, `FetchSpecificReply(replyId, IReplyKeys)`, `AcceptReply` / `RejectReply`
- Fragt Zustände ab: `GetSubmissionState`, `GetReplyState`; Vorgänge über `Cases`
 - Keine Methode, um Anträge zu empfangen — Regel 2

IOrganisationClient

`client.AsOrganisation()` · Typ A/B

- Holt Anträge ab: `FetchSubmission`, `FetchAvailableSubmissionsFromDestination` — jede mit `Report`
- Entscheidet: `AcceptSubmission` / `RejectSubmission` (signierte Events)
- Antwortet in den Vorgang: `SendReply`; sendet selbst Anträge: `SendSubmission`
 - Keine Methode, um Antworten zu lesen — Regel 3

Vier Grundregeln

Dieselben Regeln wie im Java-SDK

- **R1** Jede Partei ist ein Zustellpunkt (UUID + Typ A/B/C). Die eigene steht in `Client.DestinationId`; der Host prüft den Typ beim Start.
- **R2** Ein Antrag geht an einen Zustellpunkt — als `Guid` aus dem `Directory`, nur an A oder B.
- **R3** Eine Antwort geht in einen Vorgang (`caseId`), nie an einen Zustellpunkt — nur A/B → C.
- **R4** Schlüssel bestimmen, was Sie öffnen: `Client.DecryptionKeys` für Anträge, `IReplyKeys` für Antworten. Senden braucht keinen.

Einstieg

NUGET

Pakete

```
dotnet add package Fitko.FitConnect.Client --prerelease

# optional, je Bedarf
dotnet add package Fitko.FitConnect.VirusScanning --prerelease
dotnet add package Fitko.FitConnect.Zbp --prerelease
```

PAKET	ZWECK
<code>Fitko.FitConnect.Client</code>	Primärer Einstieg: <code>AddFitConnect()</code> , <code>IFitConnectClient</code> , Rollen-Clients, Directory
<code>Fitko.FitConnect.Core</code>	Crypto, HTTP, Validierung, <code>CallbackValidationUtil</code> — auch standalone nutzbar
<code>Fitko.FitConnect.VirusScanning</code>	ClamAV / ICAP-Integration als Zusatzpaket
<code>Fitko.FitConnect.Zbp</code>	ZBP-Adapter (Zentrales Bürgerpostfach), <code>WaitForCompletionAsync</code>
<code>Fitko.FitConnect.Examples</code>	Lauffähige Beispiele mit CLI (nur im Repo)

Kein BOM-Paket wie im Java-SDK — alle `Fitko.FitConnect.*`-Pakete werden gemeinsam versioniert.

SDK registrieren

```
// 1 – in einem Host (ASP.NET Core, Worker): IConfiguration ist schon da
builder.Services.AddFitConnect(); // Section "FitConnect"

// 2 – ohne Host
IFitConnectClient client = new ServiceCollection()
    .AddSingleton<IConfiguration>(configuration)
    .AddFitConnect()
    .BuildServiceProvider()
    .GetRequiredService<IFitConnectClient>();

// 3 – programmatisch
services.AddFitConnect(cfg =>
{
    cfg.EnvironmentTag = "TEST";
    cfg.Client = new ClientConfig { ClientId = "...", ClientSecret = "...", DestinationId =
myDestinationId };
});
```

Eine Credential-Paarung (`Client`) für Senden und Empfangen, dazu die UUID des **eigenen** Zustellpunkts. `Client.VerifyDestinationOnStartup` (Default `true`) schlägt sie beim Host-Start nach und merkt sich den Typ.

IFITCONNECTCLIENT

Alle Einstiegspunkte

```
client.AsOnlineService(); // IOnlineServiceClient – Typ C: senden, Antworten lesen
client.AsOrganisation(); // IOrganisationClient – Typ A/B: empfangen, antworten
client.Directory; // IDirectoryService – Zustellpunkte und Schlüssel
suchen

// Low-Level, nur wenn die Rollen-Clients nicht reichen:
client.SubmissionService; // ISubmissionService – beide Rollen ungefiltert
client.DestinationApiService; // IDestinationApiClient – Zustellpunkte verwalten
(Scope manage-destinations)
client.SubmissionApiService; // ISubmissionApiClient – rohe Submission-API
client.CaseApiService; // ICaseApiClient – rohe Case-API
client.ReplyApiService; // IReplyApiClient – rohe Reply-API
```

Die falsche Fassade scheitert sofort: `FitConnectConfigurationException: Destination '...' is a type C online service – call AsOnlineService() instead of AsOrganisation()`.

`IEventLogService` (geparste Event-Log-Einträge) registriert `AddFitConnect()` ebenfalls — separat injizieren.

Senden

ANTRAG BAUEN

Die Builder-Kette

```
OutgoingSubmission submission = OutgoingSubmissionBuilder.Builder() // IDestinationStep
    .WithDestinationId(destinationId) // → IServiceStep
    .WithServiceType(serviceIdentifier, serviceName) // → IMetadataStep (oder
    WithProcessMessage)
    .WithMetadataVersion(new Version(1, 5, 0)) // → IDataStep
    .WithJsonData(json, schemaUri) // → ISubmissionBuilder (oder
    WithXmlData)

    // – alles Weitere optional, Reihenfolge frei –
    .WithCaseId(existingCaseId) // Nachreichung in bestehenden
    Vorgang
    .WithAttachment(Attachment.FromFile(pdf, "application/pdf"), shouldBeChunked: false)
    .WithReplyChannel(ReplyChannel.OfFitConnect(replyPublicKey, myDestinationId))
    .WithAuthor(author)
    .WithAuthenticationInformation(authInfo)
    .WithPaymentInformation(payment)
    .WithDataSet(dataSet)
    .Build();

SentSubmission sent = await onlineService.SendSubmission(submission);
sent.SubmissionId; sent.CaseId;
```

Der Typ wandert mit: jeder Schritt gibt ein eigenes Step-Interface zurück. Was gerade nicht erlaubt ist, ist gar nicht erst sichtbar — wie bei Javas `SubmissionDataStep`. Auch eine `IOrganisationClient` darf `SendSubmission` (Verwaltung an Verwaltung), aber nicht vorverschlüsselt.

FACHDATEN

WithJsonData / WithXmlData

```
.WithJsonData(string data, Uri schemaUri)
.WithXmlData (string data, Uri schemaUri)
```

Die Metadaten-Version ist im .NET-SDK ein expliziter Pflichtschritt (`WithMetadataVersion`); Java leitet sie aus den gesetzten Feldern ab.

ATTACHMENT

Anhänge erzeugen

```
Attachment.FromBytes(bytes,
"application/pdf");
Attachment.FromFile(path,
"application/pdf");
Attachment.FromStream(stream,
"video/mp4", "movie.mp4",
shouldBeChunked: true);
Attachment.FromStreamSupplier(() =>
db.Open(id), "application/pdf",
"akte.pdf");

Attachment.Builder()
    .FromFile(path)
    .SetMimeType("application/pdf")
    .SetFileName("bescheid.pdf")

    .SetPurpose(AttachmentPurpose.Attachment)
    // Form | Attachment | Report | Data
    .SetShouldBeChunked(true)
    .Build();
```

Details zu großen Anhängen: [Große Anhänge](#).

REPLYCHANNEL

Rückkanal anbieten

```
ReplyChannel.OfFitConnect(ApiJwk encryptionKey, Guid destinationId,
List<SubmissionSchema>? supportedSchemas = null);
ReplyChannel.OfEmail();
ReplyChannel.OfEmailWithPgp();
ReplyChannel.OfDeMail();
ReplyChannel.OfElster();
ReplyChannel.OfFink();
ReplyChannel.OfIdBundDeMailbox(useStatusMonitor: true);
```

Nur `OfFitConnect(...)` macht eine Antwort über FIT-Connect möglich. Den privaten Schlüssel merken Sie sich pro Vorgang — Details: [Antworten senden und lesen](#).

VORVERSCHLÜSSELT

EncryptedOutgoingSubmission

```
// Für Architekturen, die im Browser/Frontend verschlüsseln – das SDK sieht die Klardaten nie.
EncryptedOutgoingSubmission encrypted = EncryptedOutgoingSubmission.Builder() //
    IEncryptedDestinationStep
        .SetDestination(destinationId) // → IEncryptedServiceTypeStep
        .SetServiceType(serviceIdentifier, serviceName) // → IEncryptedMetadataStep
        .SetEncryptedMetadata(jweMetadata) // → IEncryptedDataStep
        .SetEncryptedData(jweData) // →
    IEncryptedOptionalPropertiesStep
        .AddEncryptedAttachment(attachmentId, jweAttachment)
        .Build();

SentSubmission sent = await onlineService.SendEncryptedSubmission(encrypted);
```

Nur auf `IOnlineServiceClient` — wie in Java ein Typ-C-Szenario. Den öffentlichen Schlüssel für das Frontend liefert `client.Directory.GetActiveEncryptionKey(destinationId)`.

Empfangen

IORGANISATIONCLIENT

Anträge abholen und bearbeiten

```
IOrganisationClient organisation = client.AsOrganisation();

// bekannte submissionId/caseId (z. B. aus Callback)
IncomingSubmission received = await organisation.FetchSubmission(submissionId, caseId);

// oder: alle verfügbaren Submissions der eigenen Destination (Polling)
List<IncomingSubmission> available =
    await organisation.FetchAvailableSubmissionsFromDestination(myDestinationId);

if (received.Acceptable())
    await organisation.AcceptSubmission(received);
else
    await organisation.RejectSubmission(received, received.Report.AsProblems().ToList());
```

`FetchAvailableSubmissionsFromDestination` lädt direkt vollständig (entschlüsselt + validiert) — anders als Java, das erst nur IDs (`SubmissionForPickup`) liefert. Nur den Zustand ohne Entschlüsselung: `GetSubmissionState(submission)` mit einer `Submission` aus `client.SubmissionApiService.GetAvailableSubmissionsFromDestination(...)`.

INCOMINGSUBMISSION

Was drinsteht

PROPERTY/METHODE	BEDEUTUNG
SubmissionId / CaseId / DestinationId / FromDestinationId	UUIDs (Absender als FromDestinationId)
GetDataAsString() / GetDataAsBytes()	Fachdaten, entschlüsselt
GetDataSchemaUri() / GetDataMimeType()	Schema-URI / MIME- Type
Metadata	vollständiges Metadaten-Objekt
Attachments	List<Attachment>, entschlüsselt
Report / Acceptable()	Prüfbericht — false nur bei Fehlern
PublicService / ProcessMessage / Region	Adressierung des Antrags

Dieselbe Klasse wird für Submissions **und** Replies verwendet.

RECEIVERREPORT

Der Prüfbericht

```
received.Report.Acceptable(); // keine
Fehler (Warnungen zählen nicht)
received.Report.Errors; //
IReadOnlyList<ReceiveIssue>
received.Report.Warnings;
received.Report.Describe(); // eine
Zeile pro Fund + Lösungsvorschlag
received.Report.AsProblems(); // →
RejectSubmission(received, ...)
```

AcceptSubmission trotz Fehlern wirft
SubmissionNotAcceptableException. Details:
[Konzept: Der Prüfbericht](#).

Antworten

A/B → C: eine Antwort geht immer in einen Vorgang, nie an einen Zustellpunkt.

Antwort senden (Organisation)

```
// aus dem empfangenen Antrag – Vorgang und Rückkanalschlüssel werden übernommen
ReplyRequest reply = ReplyRequestBuilder.ForSubmission(received) // IMetadataStep
    .WithMetadataVersion(new Version(1, 2, 0)) // → IDataStep
    .WithJsonData(bescheidJson, schemaUri) // → IReplyBuilder
    .WithAttachment(Attachment.FromFile(pdf, "application/pdf"))
    .Build();

// aus persistierten Werten, ohne den Antrag im Speicher
ReplyRequest reply2 = ReplyRequestBuilder.ForCase(caseId) // IEncryptionStep
    .EncryptWith(replyEncryptionKey) // ApiJwk aus dem
ReplyChannel
    .WithMetadataVersion(new Version(1, 2, 0))
    .WithJsonData(bescheidJson, schemaUri)
    .Build();

SendReplyResponse sent = await organisation.SendReply(reply);
sent.ReplyId; sent.CaseId;
EventState state = await organisation.GetReplyState(sent);
```

Bei mehreren Submissions im selben Vorgang gilt der Rückkanalschlüssel der **neuesten**. Ohne FIT-Connect-Rückkanal: `FitConnectReplyException`.

Antworten lesen (OnlineService)

```
IReplyKeys keys = ReplyKeys.FromLookup(vault.Find); // Func<Guid caseId, string?
jwkJson>
IReplyKeys keys2 = ReplyKeys.Of(dictionary); // IReadOnlyDictionary<Guid, string>
IReplyKeys none = ReplyKeys.None(); // liest nie Antworten

foreach (ListedReply r in await onlineService.FetchAvailableReplies() ?? [])
{
    IncomingSubmission reply = await onlineService.FetchSpecificReply(r.ReplyId, keys);
    await onlineService.AcceptReply(reply); // oder RejectReply(reply,
problems)
}
```

Der Schlüssel gehört zum Vorgang, nicht zum Zustellpunkt — deshalb eine Nachschlagefunktion statt eines Schlüssels. Fehlt er: `FitConnectReplyException` mit Vorgang und Antwort im Text.

Cases & Eventlog

CASES

Vorgänge abfragen und anlegen

```
IOnlineServiceCases cases = onlineService.Cases; // bzw. organisation.Cases

Case? c = await cases.GetCase(caseId);
GetCasesResponse? all = await cases.GetCases(offset: 0, limit: 100);
Case? opened = await cases.CreateCase(new CreateCaseRequest(/* ... */)); // API v3,
prozessbasiert
CaseEventLogResponse? raw = await cases.GetCaseEventLog(caseId); // signierte JWT-
Strings
```

Ein Vorgang entsteht sonst implizit beim Senden der ersten Submission; Nachreichungen mit `WithCaseId(...)`.

ZUSTÄNDE

Neuester Zustand

```
EventState s1 = await onlineService.GetSubmissionState(sent); // SentSubmission
EventState s2 = await organisation.GetSubmissionState(submission); // Submission aus der
Pickup-Liste, ohne Entschlüsselung
EventState s3 = await organisation.GetReplyState(sentReply); // SendReplyResponse
EventState s4 = await onlineService.GetReplyState(listedReply); // ListedReply
```

EVENTSTATE	SCHREIBT	
Submitted	Plattform	eingereicht, nicht abgeholt
Notified	Plattform	Callback zugestellt
Forwarded	Plattform	abgeholt, nicht entschieden
Accepted / Rejected	Empfänger 	entschieden; Rejected mit Problems
Deleted / Incomplete	Plattform	entfernt / unvollständig

IEVENTLOGSERVICE

Geparste Einträge

```
// separat injizieren – wird von AddFitConnect() registriert
List<CaseEvent> log = await eventLogService.GetCaseEventLog(caseId, myDestinationId);
List<CaseEvent> one = await eventLogService.GetSubmissionEventLog(received);

foreach (CaseEvent e in log)
    Console.WriteLine($"{e.IssueTime:u} {e.EventType.State} {e.Issuer} {e.Problems?.Count
?? 0} Problems");
```

`CaseEvent`: `EventType` (mit `.State`), `Issuer`, `EventId`, `CaseId`, `SubmissionId`, `IssueTime`, `AuthTags`, `Problems`. Komfort für Sender: `WaitForCompletionAsync(sent, timeoutSeconds: 60)` aus `Fitko.FitConnect.Zbp.Extensions` pollt bis `Accepted/Rejected`.

Directory

IDIRECTORYSERVICE

Zuständige Zustellpunkte finden

```
IDirectoryService dir = client.Directory;    // unauthentifiziert

// Verwaltungsleistung + genau ein Gebietsfilter (AGS | ARS | AreaId)
IReadOnlyList<Route> routes = await dir.FindByService(leikaKey,
AreaFilter.Ars("147130000000"), offset: 0, limit: 500);
Guid destinationId = routes.Single().DestinationId;

// Prozessnachricht (API v3) – liefert vollständige Destination-Objekte
MultipleDestinations? byProcess = await dir.FindByProcessMessage(processModelId,
messageId, "DE147130000000");

// Gebiets-IDs über Namen/PLZ
AreaList? areas = await dir.FindAreas(["Leip*", "04229"], offset: 0, limit: 100);

// Aktiver öffentlicher Schlüssel (für Vorverschlüsselung)
ApiJwk key = await dir.GetActiveEncryptionKey(destinationId);
```

`AreaFilter.Ags(...)` / `Ars(...)` / `Area(...)` — der Typ erzwingt genau einen Filter. Default-Limits **500** (`FindByService`) bzw. **100** (`FindByProcessMessage`, `FindAreas`).

ROUTE

Rückgabeobjekt — bewusst schmal

```
public sealed record Route
{
    public Guid DestinationId;
    public string DestinationSignature;
    public string DestinationName;
}
```

Die Route sagt, *welcher* Zustellpunkt zuständig ist, nicht *wohin* technisch gesendet wird (das regelt die Umgebung). Mehr Felder (Adresse, Kontakt) liefert

`client.DestinationApiService.GetDestination(id)`.

Verwaltung

IDESTINATIONAPICLIENT

Zustellpunkte pflegen

```
IDestinationApiClient d = client.DestinationApiService; // Scope manage-destinations

Destination? created = await d.CreateDestination(createDestination);
Destination? one     = await d.GetDestination(destinationId);
MultipleDestinations? all = await d.ListDestinations(offset: 0, limit: 100);

await d.UpdateDestination(updatedDestination); // vollständig ersetzen
await d.PatchDestination(destinationId, patchDestination); // teilweise ändern – kein
Name-Feld
await d.DeleteDestination(destinationId);
```

`CreateDestination`, `ContactInformation` & Co. sind Primary-Constructor-Typen **ohne** parameterlosen Konstruktor — Positionsargumente statt Objektinitialisierer. Beispiele: [Zustellpunkte verwalten](#).

SCHLÜSSEL UND LIMITS

Rollover & Attachment-Limits

```
await d.AddKey(destinationId,
newPublicKey); // ApiJwk –
Rollover
ApiJwks? keys = await
d.ListKeys(destinationId, 0, 100);

DestinationAttachmentLimit? limit = await
d.GetDestinationAttachmentLimit(destinationId);
DestinationAttachmentLimit? changed =
await d.RequestAttachmentLimitChange(
destinationId, new
AttachmentLimitChangeRequest { /* Values,
RequestReason, ContactEmail */ });
```

Der aktive öffentliche Schlüssel eines fremden Zustellpunkts kommt über `client.Directory.GetActiveEncryptionKey(id)`.

ANHANGSPEICHER

Storage-Provider austauschen

```
services.AddSingleton<IStorageProvider>
(sp => new MyS3StorageProvider(/* ... */));
```

Standard ist `FileSystemStorageProvider` (System-Temp oder `Attachments.BaseDirectory`). Eigenen Speicher **vor** `AddFitConnect()` registrieren — Standard-.NET-DI, kein eigener `customization()`-Mechanismus wie in Java.

Schlüssel & Krypto

SCHLÜSSELMODELL

Konfiguration statt Aufruf-Argument

```
{
  "FitConnect": {
    "Client": {
      "DestinationId": "...",
      "DecryptionKeys": [ "...aktuell...", "...
vorherig..." ],
      "SignatureKey": "..."
    }
  }
}
```

Anders als Javas `DestinationKeys` (am Aufruf) liegen Entschlüsselungs- und Signaturschlüssel in `Client` — `DecryptionKeys` geordnet, **erster Eintrag = aktiv**. Rollover: neuen Schlüssel vorn einfügen, alten behalten, bis keine offenen Submissions mehr damit verschlüsselt sind. Der Rückkanalschlüssel dagegen ist pro Vorgang: `IReplyKeys`. Details: [Schlüssel und Rollover](#).

ICRYPTOSERVICE

Ver-/Entschlüsselung direkt nutzen

```
var crypto =
services.GetRequiredService<ICryptoService>();

string jwe =
crypto.EncryptBytes(publicKeyJson,
bytes);
byte[] plain =
crypto.DecryptBytes(privateKeyJson,
encryptedData);
byte[] plain2 =
crypto.DecryptBytes(decryptionKeysJsonList, encryptedData); // Rollover: erster
Treffer gewinnt
```

Keine Schlüsselerzeugung im SDK — Ephemeral-JWK-Paare für den Rückkanal mit `System.Security.Cryptography.RSA` + `Microsoft.IdentityModel.Tokens` erzeugen, siehe [Antworten senden und lesen](#).

CALLBACKS

Signatur verifizieren

```
using Fitko.FitConnect.Core.Callbacks;

var result = CallbackValidationUtil.ValidateCallback(
    request.Headers[CallbackHeaders.Authentication],
    request.Headers[CallbackHeaders.Timestamp],
    rawBodyBytes, callbackSecret);

if (!result.IsValid) return Results.Unauthorized(); // result.GetErrorMessage()
```

HMAC-SHA-512 über `{timestamp}.{body}` in konstanter Zeit, Zeitstempel $\pm$ `DefaultMaxAge` (5 min). Body `roh` lesen. Vollständig: [Callbacks prüfen](#).

Validierung & Metadaten

```
{
  "FitConnect": {
    "Validation": { "Metadata": true, "Data": true, "Attachments": true }
  }
}
```

```
}  
}
```

Alle drei Default `true`. Funde werfen nicht, sondern landen im `Report` der empfangenen Nachricht (`ReceiveIssue` mit Schwere, Stufe `Metadata/Attachments/Data`, Meldung und Lösungsvorschlag). Es gibt kein Auto-Reject — siehe [Konzept: Der Prüfbericht](#).

Problems (unter `Fitko.FitConnect.Client.Validation.*.Problems`) — landen im Event-Log und sind für den Absender lesbar:

GRUPPE	TYPEN
Metadata	<code>UnsupportedService</code> , <code>UnsupportedReplyChannel</code> , <code>UnsupportedMetadataSchema</code> , <code>UnsupportedDataSchema</code> , <code>MissingData</code> , <code>AttachmentsMismatch</code> , <code>IncorrectMetadataAuthenticationTag</code>
Data	<code>DataHashMismatch</code> , <code>DataJsonSyntaxViolation</code> , <code>DataXmlSyntaxViolation</code> , <code>IncorrectDataAuthenticationTag</code>
Attachments	<code>AttachmentHashMismatch</code> , <code>IncorrectAttachmentAuthenticationTag</code>

Eigene Gründe: von `Problem` (`Fitko.FitConnect.Core.Validation`) ableiten. Das Namensschema ist zu Java strukturell vergleichbar, nicht 1:1 identisch.

Virenprüfung: Paket `Fitko.FitConnect.VirusScanning` hinzufügen und per `AddFitConnectVirusScanning()` registrieren (`VirusScannerMode`: `ClamAVDaemon` · `ClamAVProcess` · `Icap` · `NoOp`). Details: [Virens Scanner anbinden](#).

Konfiguration

```
{  
  "FitConnect": {  
    "EnvironmentTag": "TEST",  
    "Client": {  
      "ClientId": "...", "ClientSecret": "...", "DestinationId": "...",  
      "DecryptionKeys": [ "... " ], "SignatureKey": "...", "VerifyDestinationOnStartup":  
true  
    },  
    "Http": {  
      "Timeouts": { "Read": 30, "Write": 30, "Connection": 10 },  
      "Proxy": { "Host": "", "Port": 0 },  
      "Retry": { "AllowRetries": true, "MaxRetryCount": 5, "InitialDelayInMs": 500,  
        "RetryableStatusCodes": [408, 429, 500, 502, 503, 504] }  
    },  
    "Attachments": { "ChunkAllAttachments": false, "ChunkSizeInMb": 10,  
"BaseDirectory": "" },  
    "Validation": { "Metadata": true, "Data": true, "Attachments": true }  
  }  
}
```

Pflicht: `EnvironmentTag`, `Client.ClientId`, `Client.ClientSecret`, `Client.DestinationId`.
`Http.Retry.AllowRetries` ist standardmäßig **false** — für Produktion aktivieren.
`Attachments.ChunkSizeInMb` hat keinen sinnvollen Default und muss bei aktiviertem Chunking gesetzt sein.

	AUTH / API	INSECURE KEYS
TEST	<code>*.fit-connect.fitko.dev</code>	✓
STAGE	<code>stage.fit-connect.fitko.net</code>	✗
PROD	<code>prod.fit-connect.fitko.net</code>	✗

STAGE nutzt für Routing dieselbe URL wie PROD — es gibt keine eigene STAGE-Routing-API. Eigene Umgebung: `EnvironmentTag: "CUSTOM"` + `CustomEnvironment { TokenUrl, SubmissionUrls, RoutingUrl, SspUrl, DestinationUrl }` (alle Pflicht). Weitere Tags: `LOCAL`, `CI`.

WERT	BEDEUTUNG
13 MB	max. Inline-Fachdaten, darüber automatisch als Anhang
500	Default-Limit für <code>ListDestinations</code> , <code>GetAvailableSubmissionsFromDestination</code> , <code>FindByService</code>
100	Default-Limit für <code>GetCases</code> , <code>FindAreas</code> , <code>FindByProcessMessage</code>

Vollständige Property-Referenz: [Konfiguration](#).

Fehlerbehandlung

Alle SDK-Fehler erben von `FitConnectException` (`Fitko.FitConnect.Core.Exceptions`):

EXCEPTION	AUSLÖSER
<code>FitConnectConfigurationException</code>	Credentials fehlen, Umgebung unbekannt, <code>DestinationId</code> existiert nicht oder passt nicht zur Fassade
<code>FitConnectAuthorizationException</code>	OAuth-Token nicht erhalten oder abgelehnt (401/403)
<code>FitConnectCryptoException</code>	Ver-/Entschlüsselung, Envelope, Schlüsselmaterial
<code>FitConnectSchemaException</code>	Schema nicht auflösbar oder nicht unterstützt

EXCEPTION	AUSLÖSER
<code>FitConnectSenderException</code>	Sendeseitig inkonsistent (Ziel, Case-Auswahl, Metadaten-Version)
<code>FitConnectSubscriberException</code>	Empfangsseitig inkonsistent
<code>FitConnectReplyException</code>	Kein FIT-Connect-Rückkanal / Rückkanalschlüssel fehlt
<code>SubmissionNotAcceptableException</code>	<code>AcceptSubmission</code> / <code>AcceptReply</code> trotz Fehlern im <code>Report</code>
<code>FitConnectAttachmentException</code> / <code>FitConnectAttachmentDecryptionException</code>	Upload, Chunking, Download, Entschlüsselung eines Anhangs
<code>FitConnectRoutingException</code>	Directory-Suche fehlgeschlagen oder ungültige Filter
<code>FitConnectEventParsingException</code>	Event-Log-Eintrag nicht lesbar
<code>FitConnectMappingException</code>	JSON ↔ Modell-Mapping
<code>EventCreationException</code>	Ausgehendes Event konnte nicht gebaut/signiert werden (z. B. <code>SignatureKey</code> fehlt)
<code>AuthenticationTagException</code>	Authentifizierungs-Tag ungültig
<code>FitConnectStorageException</code> (<code>Core.Util</code>)	Lesen/Schreiben im Anhangspeicher
<code>RestApiException</code> (<code>Core.Http.Exceptions</code>)	Transportfehler ohne speziellere Abbildung; nicht von <code>FitConnectException</code> abgeleitet

Vollständige Tabelle mit Handlungsempfehlungen: [Exceptions](#).

Beispiele

```
dotnet run --project Fitko.FitConnect.Examples -- submission # senden + abholen
dotnet run --project Fitko.FitConnect.Examples -- reply      # BiDiKo komplett
(erzeugt Reply-JWks)
dotnet run --project Fitko.FitConnect.Examples -- process    # v3:
Prozessnachricht, Case anlegen
dotnet run --project Fitko.FitConnect.Examples -- zbp        # ZBP-Nachricht über
den Adapter
dotnet run --project Fitko.FitConnect.Examples -- virusscanning # mit Virensan
```

```
dotnet run --project Fitko.FitConnect.Examples -- callback # Callback offline  
verifizieren
```

Kein vorgebautes Executable wie `fit-connect-cli.jar` — Sie bauen `Fitko.FitConnect.Examples` selbst; die `appsettings.*.json` mit Credentials sind nicht im Repo enthalten. Details: [Kommandozeile & Beispielprojekte](#).

Fallstricke

MODELLFEHLER

Falsche Erwartungen ans Modell

- Die falsche Fassade holen: `AsOrganisation()` für einen Typ-C-Zustellpunkt scheitert beim Host-Start mit `FitConnectConfigurationException` — der Typ ist bei der Anlage fest.
- `client.SubmissionService` statt der Rollen-Clients verwenden — es funktioniert, umgeht aber den Rollenschnitt; nehmen Sie es nur, wenn beide Rollen in einem Prozess laufen müssen.
- Eine Antwort an einen **Zustellpunkt** adressieren zu wollen — Antworten gehen in den Vorgang (`ForSubmission(received)` / `ForCase(caseId)`).
- `DestinationKeys` als Typ suchen — Entschlüsselungs- und Signaturschlüssel liegen in `FitConnect.Client`; nur der Rückkanalschlüssel ist ein Aufruf-Argument (`IReplyKeys`).
- `new CreateDestination { X = ... }` schreiben — Primary-Constructor-Typ ohne Objektinitialisierer.

HANDHABUNG

Typische Laufzeitfehler

- Zwei Gebietsfilter kombinieren wollen — `AreaFilter` nimmt genau einen.
- `ReplyChannel.OfFitConnect(key, destinationId)` mit der falschen Destination: es muss **Ihre** sein, dorthin wird geantwortet.
- `UpdateDestination` statt `PatchDestination` für Teiländerungen — leert ausgelassene Felder.
- `Attachments.ChunkSizeInMb` nicht setzen, aber Chunking aktivieren — erzeugt 0-MB-Chunks statt eines Startfehlers.
- Bei mTLS/ZBP eine der drei Zertifikatsdateien (`ClientCertPath`/`ClientPrivateKeyPath`/`CaCertPath`) vergessen — `AddFitConnectZbp()` wirft erst beim Start.

PRODUKTION

Vor dem Go-Live prüfen

- `Http.Retry.AllowRetries` aktivieren — Default ist `false`.
- Schlüssel und Secrets nicht in `appsettings.json` einchecken — User Secrets lokal, Vault-Provider im Betrieb.
- Beim Schlüsselwechsel alten und neuen Schlüssel gleichzeitig in `Client.DecryptionKeys` halten (neu zuerst).
- `Attachments.BaseDirectory` setzen — der Default liegt im temporären Systemverzeichnis.
- `VerifyDestinationOnStartup` braucht Netz beim Start; für Offline-Builds auf `false`, dann scheitert eine falsche Fassade erst serverseitig.
- Callback-Body roh lesen, `submissionId`/`caseId` immer mitloggen.

Ausführlich erklärte Schritt-für-Schritt-Rezepte finden Sie unter [Antrag senden](#), [Anträge abholen und prüfen](#) und den weiteren Seiten unter **Rezepte** in der Seitenleiste. Änderungen zwischen SDK-Ständen: [Changelog & Migration](#).



[Impressum](#) [Datenschutzerklärung](#) [Newsletter](#)



Diese Seite ist Teil des [Föderalen Entwicklungsportals](#). Verantwortlich für die Inhalte der Seite sind die jeweiligen Autoren. Wenn nicht anders vermerkt, sind die Inhalte dieser Webseite lizenziert unter der [Creative Commons Namensnennung 4.0 International Public License \(CC BY 4.0\)](#). Die technische Infrastruktur wird betrieben durch die FITKO.