

Spickzettel (Java)

👤 Rolle

Verwaltungssystem

Onlinedienst

📦 SDK · Modul

Java · sdk-client

✓ Voraussetzungen

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

STAND

4.0.0-rc.1

Java 21+

dev.fitko.fitconnect:sdk-client

API v3

TL;DR – Alle Kernbausteine des Java-SDKs auf einer Seite: Rollen, Adressieren, Senden, Empfangen, Antworten, Cases, Routing, Verwaltung, Krypto, Validierung, Konfiguration, Fehlerbehandlung, CLI und die häufigsten Fallstricke.

📌 STAND DIESER SEITE

Diese Seite beschreibt das Objektmodell von **4.0.0-rc.1** (Branch `main`), wie es der aktuelle Java-SDK-Stand tatsächlich verwendet: `Organisation/OnlineService` als Rollen, `Participant/Addressing` zur Adressierung, `OutgoingSubmission.to(...)` als Builder. Die Rezeptseiten (siehe **Java & .NET SDK** → **Rezepte** in der Seitenleiste) sind auf denselben Stand synchronisiert.

Organisation A/B

`sdk.organisation(destinationId, keys)`

- Empfängt **Anträge** und öffnet sie mit `DestinationKeys`
- Nimmt an oder weist zurück (`accept` / `reject`)
- Sendet **Anträge** an andere A/B-Destinations
- Sendet **Antworten** in Cases von C-Destinations
- Empfängt keine Antworten — das ist kein Bug (R3)

OnlineService C

`sdk.onlineService(destinationId, replyKeys)`

- Sendet **Anträge** an A/B-Destinations
- Sendet auch **vorverschlüsselt** (`EncryptedOutgoingSubmission`)
- Empfängt **Antworten** und öffnet sie mit `ReplyKeys`
- Nimmt Antworten an oder weist sie zurück
- Empfängt keine Anträge — nur A/B sind Empfänger (R2)

Die vier Protokollregeln

Sie erklären fast jede API-Entscheidung

- **R1** Jede Partei ist eine Destination: UUID + Typ A/B/C.
- **R2** Ein Antrag geht an eine Destination — nur A oder B.
- **R3** Eine Antwort geht an einen Case, nie an eine Destination.
- **R4** Schlüssel bestimmen, was man öffnen kann, nie was man senden darf.

Einstieg

MAVEN

Abhängigkeit

```
<dependency>
  <groupId>dev.fitko.fitconnect</groupId>
  <artifactId>sdk-client</artifactId>
  <version>4.0.0-rc.1</version>
</dependency>
```

Mehrere Module? Einmal die BOM `sdk-bom` in `dependencyManagement` importieren, danach überall ohne `<version>` deklarieren.

MODUL	ZWECK
<code>sdk-client</code>	High-Level-Client, alles Protokoll inklusive
<code>core</code>	Krypto, HTTP, Schema-Validierung
<code>rest-clients</code>	Rohe REST-Zugriffe, OAuth, API-Modelle
<code>virus-scanner</code>	ICAP, ClamAV-Daemon oder <code>clamscan</code>
<code>zbp-client</code>	Zentrales Bürgerpostfach (BundID)
<code>cli</code>	<code>fit-connect-cli.jar</code> , nicht auf Maven Central

BOOTSTRAP

SDK-Instanz erzeugen

```
// 1 – Builder (üblich)
FitConnectSdk sdk = FitConnectSdk.fromConfigBuilder()
    .credentials("client-id", "client-secret")
    .environment(FitConnectEnvironment.TEST)
    .settings(SdkSettings.withDefaultSettings())
    .build();

// 2 – aus YAML
FitConnectSdk sdk = FitConnectSdk.fromConfigYaml(Path.of("config.yml"));
```

Eine Credential-Paarung für Senden und Empfangen — die alte Trennung in `senderConfig`/`subscriberConfig` (aus v3) gibt es nicht mehr.

EINSTIEGSPUNKTE

Vier Wege ins SDK

```
Organisation amt = sdk.organisation(destId, keys); // A/B
Organisation amt = sdk.organisation(destId); // A/B, nur senden
OnlineService svc = sdk.onlineService(destId, replyKeys); // C
OnlineService svc = sdk.onlineService(destId); // C, nur senden

Directory dir = sdk.directory(); // Suche, unauthentifiziert
Management mgm = sdk.manage(); // Destinations + Anhänge
```

Die Destination, als die Sie handeln, steht **nicht** in der Konfiguration — sie gehört an die Aufrufstelle, zusammen mit ihrem Schlüsselmaterial.

ORGANISATION · TYP A/B

Alle Methoden

METHODE	RÜCKGABE
<code>awaitingSubmissions()</code>	<code>List<SubmissionForPickup></code>
<code>awaitingSubmissions(offset, limit)</code>	<code>SubmissionsForPickup</code>
<code>receive(SubmissionForPickup)</code>	<code>ReceivedSubmission</code>
<code>receive(UUID submissionId)</code>	<code>ReceivedSubmission</code>
<code>accept(received, Problem...)</code>	<code>void</code>
<code>reject(received, problems)</code>	<code>void</code>
<code>reject(waiting, problems)</code>	<code>void</code> — ohne Entschlüsseln
<code>send(OutgoingSubmission)</code>	<code>SentSubmission</code>
<code>send(OutgoingReply)</code>	<code>SentReply</code>
<code>cases()</code>	<code>OrganisationCases</code>

Alle Methoden

METHODE	RÜCKGABE
<code>send(OutgoingSubmission)</code>	<code>SentSubmission</code>
<code>send(EncryptedOutgoingSubmission)</code>	<code>SentSubmission</code>
<code>awaitingReplies()</code>	<code>List<ReplyForPickup></code>
<code>awaitingReplies(offset, limit)</code>	<code>RepliesForPickup</code>
<code>receive(ReplyForPickup)</code>	<code>ReceivedReply</code>
<code>accept(received, Problem...)</code>	<code>void</code>
<code>reject(received, problems)</code>	<code>void</code>
<code>cases()</code>	<code>OnlineServiceCases</code>

Acht Mitglieder, keine Unter-Objekte: einen Antrag zu senden soll genau eine Zeile kosten.

Empfänger adressieren

PARTICIPANT

Wer bekommt den Antrag?

```
// Genau eine Factory – beide Argumente
// Pflicht
Participant amt =
    Participant.of(destinationId,
addressing);

amt.destinationId(); // UUID
amt.addressing();    // Addressing, nie
// null
```

Nur für Anträge. Eine Antwort adressiert nie einen `Participant` — der Case ist die Adresse.

ADDRESSING

Leistung oder Prozessnachricht

```
// Nach Verwaltungsleistung (LeiKa/FIM)
Addressing.toService(serviceIdentifier,
serviceName);

// Nach Prozessnachricht
// (Prozesskommunikation)
Addressing.toProcessMessage(processModelId,
messageId);
```

Versiegelte Schnittstelle mit zwei Varianten:

`Addressing.ByService` und `Addressing.ByProcessMessage`. `region()` ist optional (ARS/AGS). **Prozessadressierte Anträge brauchen einen Case:** vorher `inCase(...)`, `openingCase(...)` oder `openingCaseAt(...)` setzen, sonst wirft `build()`.

DIRECTORY

Zuständige Stelle suchen

```
// jede Suche unauthentifiziert
Directory dir = sdk.directory();

Participant p = dir.forService(serviceId, name, ars);
List<Participant> ps =
    dir.allForService(serviceId, name, ars);
EncryptionKey k =
    dir.activeEncryptionKeyOf(destinationId);
```

Die Suchergebnisse sind bereits fertig adressierte `Participant`-Objekte — direkt in `OutgoingSubmission.to(...)` weiterreichbar.

Senden

ANTRAG BAUEN

Die Builder-Kette

```
SentSubmission sent = svc.send(
    OutgoingSubmission.to(amt)                                // Participant – Pflicht

    // – Case (optional bei Leistungs-, Pflicht bei Prozessadressierung) –
    .inCase(existingCaseId)                                    // bestehendem Case beitreten

    // – Fachdaten – Pflicht, schaltet die optionalen Setter frei –
    .setData(SubmissionData.json(json, schemaUri))

    // – alles Weitere optional, Reihenfolge frei –
    .addAttachment(Attachment.fromFile(pdf, "application/pdf"))
    .setReplyChannel(ReplyChannels.fitConnect(replyPubKey, standards))
    .setAuthor(author)                                         // Metadaten 2.x/3.x
    .setDataSets(dataSets)                                     // Metadaten 2.x/3.x
    .build());

sent.submissionId(); sent.caseId();
```

Der Typ wandert mit: `to()` liefert einen `SubmissionDataStep`, `setData()` einen `SubmissionOptionalPropertiesStep`. Was gerade nicht erlaubt ist, ist gar nicht erst sichtbar.

SUBMISSIONDATA

Fachdaten verpacken

```
SubmissionData.json(String data, URI
schemaUri);
SubmissionData.json(byte[] data, URI
schemaUri);
SubmissionData.xml (String data, URI
schemaUri);
SubmissionData.xml (byte[] data, URI
schemaUri);
```

Ein Wert bündelt MIME-Type, Schema-URI und Bytes. **Automatik:** Daten über **13 MB** werden transparent als Anhang platziert statt inline eingebettet.

ATTACHMENT

Anhänge erzeugen

```
Attachment.fromBytes(bytes,
"application/pdf");
Attachment.fromFile(
    Path.of("bescheid.pdf"),
    "application/pdf");

Attachment.builder()
    .fromFile(path)
    // FORM | ATTACHMENT | REPORT | DATA
    .purpose(Purpose.ATTACHMENT)
    .shouldBeChunked(true) //
    // gestückelt hochladen
    .build();
```

Ohne `fileName` wird eine UUID vergeben, ohne `purpose` gilt `ATTACHMENT`.

REPLYCHANNELS

Rückkanal anbieten

```
ReplyChannels.fitConnect(jwkOrEncryptionKey, processStandards);
ReplyChannels.email(address);
ReplyChannels.emailWithPgp(address, pgpPublicKey);
ReplyChannels.deMail(address);
ReplyChannels.elster(accountId, deliveryTicket, reference);
```

Nur `fitConnect(...)` macht eine Antwort über FIT-Connect möglich. Ohne ihn wirft `OutgoingReply.answerIng(received)` — und der Antrag bleibt einseitig.

VORVERSCHLÜSSELT · NUR TYP C

EncryptedOutgoingSubmission

```
// Für Architekturen, die im Browser/Frontend verschlüsseln:
// das SDK sieht die Klardaten nie.
SentSubmission sent = svc.send(
    EncryptedOutgoingSubmission.to(amt)
        .openingCase(serviceIdentifier, serviceName)
        .setEncryptedData(jweCompact, Sha512Hash.ofHex(hex), schemaUri,
        MimeTypes.APPLICATION_JSON)
        .addEncryptedAttachment(new EncryptedAttachmentMetadata(
            attachmentId, jweContent, hash, "antrag.pdf",
            "application/pdf", Purpose.ATTACHMENT, "Scan"))
        .setReplyChannel(channel)
        .build());
```

Gleiche optionale Setter wie beim normalen Antrag. `Organisation` hat diese Überladung bewusst **nicht**: vorverschlüsseltes Senden ist ein Typ-C-Szenario.

Empfangen

ORGANISATION · TYP A/B

Anträge abholen und bearbeiten

```
Organisation amt = sdk.organisation(MY_DEST_ID, keys);

for (SubmissionForPickup waiting : amt.awaitingSubmissions()) {
    ReceivedSubmission received = amt.receive(waiting); // entschlüsselt + validiert

    if (received.acceptable()) { // kurz für report().acceptable() – kein Auto-
Reject mehr
        amt.accept(received); // Quittung ins Event-Log
    } else {
        System.out.println(received.report().describe());
        amt.reject(received, received.report().asProblems());
    }
}

// Zusätzliche eigene Problems statt/neben dem Report:
amt.reject(received, List.of(new DataSchemaViolation("Feld X fehlt")));

// Zurückweisen ohne Entschlüsseln (z. B. Schlüssel fehlt)
amt.reject(waiting, List.of(new MetadataEncryptionKeyIssue("kid-2024-01")));
```

`accept` nimmt zusätzlich Probleme als Varargs entgegen — Annahme trotz Beanstandung ist damit möglich, auch wenn `acceptable()` bereits `true` ist. Ruft man `accept` auf, während `acceptable()` `false` liefert, wirft es `SubmissionNotAcceptableException`.

RECEIVEDSUBMISSION

Was drinsteht

METHODE	BEDEUTUNG
<code>id()</code> / <code>caseId()</code>	UUIDs von Transfer und Vorgang
<code>getDataAsString()</code>	Fachdaten als UTF-8-Text
<code>getAttachments()</code>	<code>List<Attachment></code> , entschlüsselt
<code>addressing()</code>	<code>ByService</code> oder <code>ByProcessMessage</code>
<code>replyChannel()</code>	<code>Optional<JWK></code> — Antwortschlüssel
<code>senderDestinationId()</code>	UUID, rein informativ

ONLINESERVICE · TYP C

Antworten abholen

```
OnlineService svc =
sdk.onlineService(MY_C_ID, replyKeys);

for (ReplyForPickup waiting :
svc.awaitingReplies()) {
    ReceivedReply reply =
    svc.receive(waiting);

    if (reply.report().acceptable()) {
        svc.accept(reply);
    } else {
        svc.reject(reply,
reply.report().asProblems());
    }
}
```

METHODE	BEDEUTUNG
<code>id()</code> / <code>caseId()</code>	Antwort- und Case-UUID
<code>getDataAsString()</code>	Antwortdaten als Text
<code>getAttachments()</code>	Anhänge der Antwort
<code>senderDestinationId()</code>	antwortende Stelle

Antworten

A/B → C: eine Antwort geht immer an einen Case, nie an eine Destination.

OutgoingReply

```
// Weg 1 – direkt aus dem empfangenen Antrag (nimmt Case + Schlüssel von dort)
SentReply sent = amt.send(
    OutgoingReply.answering(received)
        .setData(SubmissionData.json(bescheid, schemaUri)) // Pflicht
        .addAttachment(Attachment.fromFile(pdf, "application/pdf"))
        .forProcessMessage(processModelId, messageId) // Prozesskommunikation
        .preferMetadataVersion(MetadataVersion.V3)
        .build());

// Weg 2 – später, ohne den Antrag noch im Speicher zu haben
OutgoingReply.answering(caseId, replyEncryptionKeyJwk).setData(...).build();
```

Wichtig: `answering(received)` wirft eine `FitConnectReplyException`, wenn der Antrag **keinen FIT-Connect-Rückkanal** angeboten hat. Genau so setzt der Code R3 durch: A/B-Absender bieten keinen an, also ist A/B → A/B einseitig.

SCHLÜSSEL FÜR ANTWORTEN

ReplyKeys

```
// Funktionales Interface: Case-ID → Schlüssel
ReplyKeys keys =
    ReplyKeys.of(Map.of(caseId, privateJwk));
ReplyKeys keys = vault.lookupReplyKey; // Methodenreferenz
ReplyKeys keys = ReplyKeys.none(); // nur senden
```

Passend zu R4: `ReplyKeys` öffnen Antworten auf selbst gesendete Anträge (nur C), `DestinationKeys` öffnen an Sie adressierte Anträge (nur A/B). **Keine Destination hält je beides.**

Cases & Transferlog

CASES

Vorgänge

```
List<Case> all = cases.list();

// Case eröffnen
Case c = cases.open(publicService);
// bei fremder Destination
Case c = cases.open(publicService,
atDestinationId);

// ganzer Vorgang
List<CaseEvent> events =
cases.events(caseId);
```

`events(caseId)` liefert das Log des *Vorgangs*,
`logOf(nachricht)` das Log eines *einzelnen*
Transfers. Die Namen sind absichtlich
verschieden.

TRANSFERLOG

Status einer Nachricht

```
TransferLog log =
cases.logOf(sentSubmission);

// nach issueTime, wirft bei leerem Log
CaseEvent neuest = log.latest();
EventState state = neuest.state();
```

EVENTSTATE	
SUBMITTED	eingereicht
ACCEPTED	angenommen
REJECTED	zurückgewiesen
FORWARDED	weitergeleitet
NOTIFIED	benachrichtigt
DELETED	gelöscht
INCOMPLETE	unvollständig

SICHTBARKEIT

Wer sieht welches logOf

ARGUMENT	A/B	C
SentSubmission	ja	ja
SubmissionForPickup	ja	—
SentReply	ja	—
ReplyForPickup	—	ja

Deshalb `OrganisationCases` und `OnlineServiceCases` statt eines gemeinsamen breiten Typs.

Routing

ROUTING

Zuständige Destinations finden

```
Routing rt = sdk.directory().routing();

// Suche: LeiKa-Schlüssel + genau EIN Gebietskriterium
List<Route> r = rt.routes(DestinationSearch.withArs(leikaKey, ars));
List<Route> r = rt.routes(DestinationSearch.withAgs(leikaKey, ags));
List<Route> r = rt.routes(DestinationSearch.withAreaId(leikaKey, areaId));
```

FELD	MUSTER	HINWEIS
<code>leikaKey</code>	99 + 12 Ziffern	Pflicht
<code>ars</code>	2/3/5/9/12 Ziffern	Regionalschlüssel
<code>ags</code>	2/3/5/8 Ziffern	Gemeindeschlüssel
<code>areaId</code>	Ziffern	Gebiets-ID

Genau eines von `ars`/`ags`/`areaId` setzen — keines oder mehrere werfen sofort im Konstruktor. Default-Limit 500.

Verwaltung

MANAGEMENT

Destinations pflegen

```
DestinationClient d =
sdk.manage().destinations();

// vollständig ersetzen
d.updateDestination(id,
updateDestination);
// teilweise ändern
d.patchDestination(id, destinationPatch);
```

Patch statt Update, wenn nur einzelne Felder geändert werden: bei `DestinationPatch` werden ausgelassene Felder nicht mitgesendet und damit serverseitig nicht geleert.

MANAGEMENT

Schlüssel und Limits

```
d.addKeyToDestination(id, publicKey);
// Rollover
DestinationLimits l =
d.getAttachmentLimits(id);
d.requestLimitChange(
    id, LimitDirection.SEND, limits,
    grund);
```

Limits gibt es getrennt für Sende- und Empfangsrichtung; ohne `LimitDirection` gilt die Standardrichtung.

MANAGEMENT

Anhangspeicher

```
AttachmentManager a = sdk.manage().attachments();

// InputStream
a.loadAttachment(submissionId, attachmentId);
a.getTotalSize(); a.getFileCount();
```

Standard ist der lokale `FileSystemStorageProvider`. Eigenen Speicher (S3, Datenbank) über `AttachmentStorageProvider` implementieren und via `customization().withCustomStorageProvider(...)` einhängen.

Schlüssel & Krypto

DESTINATIONKEYS

Schlüssel einer A/B-Destination

```
new DestinationKeys(decryptionJwk,
signatureJwk);

// Rollover
new DestinationKeys(List.of(alt, neu),
signatureJwk);
// nur senden
DestinationKeys.none();
```

Der Signaturschlüssel signiert die Event-Log-Einträge (JWS).

SCHLÜSSEL ERZEUGEN

Testschlüssel & Public Keys

```
// RSA-4096-Paar für Verschlüsselung
(RSA-OAEP-256)
JWKPair pair =
EncryptionKeyPairGenerator.generate();

EncryptionKey k =

sdk.directory().activeEncryptionKeyOf(destinationId);
```

Wichtig: Produktivschlüssel gehören in ein HSM oder einen Vault, nicht neben die Credentials in eine Datei.

CALLBACKS

Signatur verifizieren

```
ValidationResult r = CallbackValidationUtil.validateCallback(
    hmacHeader, timestampInSec, httpBody, callbackSecret);

if (!r.isValid()) { /* 401 */ }
```

Prüft HMAC und Zeitstempel des von FIT-Connect gesendeten Callbacks, bevor der Prozess neue Nachrichten abholt.

Validierung & Metadaten

```
ValidationConfig.builder()
    .validateMetadata(true) // Schema, Auth-Tags, Service, Rückkanal
    .validateData(true)    // Hash, Syntax, JSON-Schema
```

```
.validateAttachments(true) // Hash, Auth-Tag, Inhalt
.build();
// kein enableAutoReject mehr (seit 4.0.0-rc.1) – Befunde landen im ReceiveReport,
// siehe „Empfangen“ oben bzw. Validierungsdiagnose statt Auto-Reject
```

GESETZTE FELDER	VERSION
<code>authenticationInformation</code>	V1 (1.6.0)
<code>author</code> / <code>dataSets</code> + Root-Rückkanal	V2 (2.1.0)
sonst	V3 (3.0.0)

VORSICHT

`authenticationInformation` und `author/dataSets` gleichzeitig zu setzen wirft eine `FitConnectSenderException`: 1.x und 2.x schließen sich aus.

Problems (Auswahl, aus `rest.model.event.problems`) — landen im Event-Log und sind für den Absender lesbar:

GRUPPE	BEISPIELE
metadata	<code>MetadataSchemaViolation</code> , <code>UnsupportedService</code> , <code>UnsupportedReplyChannel</code>
data	<code>DataHashMismatch</code> , <code>DataSchemaViolation</code> , <code>DataEncryptionKeyIssue</code>
attachment	<code>AttachmentHashMismatch</code> , <code>MissingAttachment</code> , <code>AttachmentEncryptionKeyIssue</code>
security	<code>MalwareDetected</code>

Virenschutz: Modul `virus-scanner` hinzufügen — das SDK scannt dann Fachdaten, Metadaten und Anhänge (`CLAMAV_DAEMON` · `CLAMAV_PROCESS` · `ICAP` · `NO_OP`). Details siehe [Virens Scanner anbinden](#).

Konfiguration

```
credentials:
  clientId: "your-client-id"
  clientSecret: "your-client-secret"

environment: "TEST" # TEST | STAGE | PROD

sdkSettings:
  httpConfig:
    timeoutConfig: { readTimeout: 30, writeTimeout: 30, connectionTimeout: 30,
callTimeout: 60 }
    retryConfig:
```

```

allowRetries: true
retryableStatusCodes: [408, 429, 500, 502, 503, 504]

attachmentChunkingConfig:
  chunkAllAttachments: false
  attachmentStoragePath: "/tmp/fit-connect-attachments"

validationConfig:
  validateMetadata: true

```

Unbekannte Felder werden ignoriert, fehlende mit Defaults gefüllt.

	AUTH / API	UNSICHERE KEYS
TEST	*.fit-connect.fitko.dev	erlaubt
STAGE	stage.fit-connect.fitko.net	nein
PROD	prod.fit-connect.fitko.net	nein

Ohne Angabe gilt `TEST`. STAGE nutzt die Produktiv-Routing-API — es gibt keine eigene.

WERT	BEDEUTUNG
13 MB	max. Inline-Fachdaten, darüber automatisch als Anhang
500	Default-Limit für Cases, Replies, Submissions, Routing
3	parallele Anhang-Streams
10 MB	Chunk-Größe beim gestückelten Upload

Fehlerbehandlung

Alle erben von `FitConnectException` (unchecked).

EXCEPTION	AUSLÖSER
<code>FitConnectConfigurationException</code>	Fehlende Credentials, kaputte YAML, unbekannte Destination
<code>FitConnectAuthorizationException</code>	OAuth-Token abgelehnt, fehlender Scope
<code>FitConnectCryptoException</code>	Ver-/Entschlüsselung, Signatur, ungeeignetes Schlüsselmaterial

BEFEHL	ZWECK
<code>doctor [<destination-id>]</code>	Selbst sdk.c

Ohne `-c` wird `config.yml` im Arbeitsverzeichnis gesucht; sie braucht zusätzlich den CLI-eigenen `identity`-Block (Destination + Schlüsselpfade). Ergebnisse gehen nach `stdout` (`--json` maschinenlesbar), Exit-Codes 0/1/2/3. Details: [Kommandozeile](#) & [Beispielprojekte](#).

Fallstricke

MODELLFEHLER

Falsche Erwartungen ans Modell

- Eine Antwort an eine **Destination** adressieren zu wollen — Antworten gehen an den Case.
- Erwarten, dass eine `Organisation` Antworten empfängt — tut sie nicht, das ist korrekt (R3).
- Einen Antrag an eine C-Destination senden — nur A und B sind Empfänger (R2).
- Eine Destination mit `DestinationKeys` und `ReplyKeys` ausstatten — die Identitäten sind disjunkt.

HANDHABUNG

Typische Laufzeitfehler

- `OutgoingReply.answer(received)` ohne angebotenen FIT-Connect-Rückkanal — wirft.
- Prozessadressierter Antrag ohne Case — `build()` wirft.
- `log.latest()` auf einem leeren Event-Log — wirft; vorher `entries()` prüfen.
- `updateDestination` statt `patchDestination` für Teiländerungen — leert Felder.

PRODUKTION

Vor dem Go-Live prüfen

- Diagnosebericht **immer** vor `accept` prüfen (`received.acceptable()`) — es gibt kein Auto-Reject mehr, das ungültige Nachrichten von selbst aussortiert.
- Private Schlüssel nicht neben die Credentials in die YAML legen — sie sind Argumente.
- Beim Schlüsselwechsel alten und neuen Schlüssel gleichzeitig in `DestinationKeys` halten.
- `attachmentStoragePath` setzen — der Default liegt im temporären Verzeichnis.

Ausführlich erklärte Schritt-für-Schritt-Rezepte finden Sie unter [Submission senden](#), [Submission empfangen](#) und den weiteren Seiten unter **Java & .NET SDK** → **Rezepte** in der Seitenleiste. Änderungen zwischen SDK-Versionen: [Changelog](#).



FIT-CONNECT



EIN PRODUKT DES IT-PLANUNGSRATS

[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.