

POS Integration Guide

Developer reference for integrating a POS with the XPOS ESDC local service (port 8888): endpoints, request/response schemas, errors and the legacy V2 API.

- [Overview & Integration Flow](#)
- [Status & Session](#)
- [Signing an Invoice](#)
- [Errors, Response Codes & Validation](#)
- [History & Reports](#)
- [Legacy V2 API](#)

Overview & Integration Flow

XPOS ESDC exposes a local HTTP service that a Point of Sale (the built-in POS, or your own POS/ERP) uses to fiscalise sales. This guide is the integration reference for developers building against that service.

Base URL

The service listens on port **8888** on all interfaces of the machine running XPOS ESDC:

```
http://localhost:8888      (POS on the same PC)
http://<esdc-host>:8888    (POS elsewhere on the LAN)
```

Two API surfaces are available:

- **v3** — the current API, under `/api/v3`. Use this for new integrations.
- **Legacy V2** — `/api/attention`, `/api/status`, `/api/sign`, for compatibility with older POS integrations. See the "Legacy V2 API" page.

Interactive reference (Swagger)

The running service publishes a live, always-current OpenAPI document and an interactive UI:

```
http://localhost:8888/swagger/index.html
```

The Swagger schema is authoritative for exact field names and shapes — this guide explains the flow and the important fields; Swagger is the definitive contract for the build you are integrating with.

The integration flow

A POS follows this sequence:

1. **Check availability** — `GET /api/v3/attention`. Confirms the SDC service is up and a reader is present.
2. **Read status** — `GET /api/v3/status`. Tells you whether a PIN is required (`isPinRequired`), whether an audit is required, the SDC date/time, the device identity, and the active tax

rates.

3. **Verify PIN (once per session)** — if `isPinRequired` is true, `POST /api/v3/pin` with the operator's PIN. The SDC caches it in memory and signs subsequent invoices without it. The PIN is cleared on restart, so this step repeats after the app restarts.
4. **Sign invoices** — `POST /api/v3/invoices` for each sale. The response contains the verification URL, QR code, journal (receipt text) and signature.
5. **Print / store** — render the receipt from the response, or fetch the rendered PDF later from history.

Idempotency (RequestId)

On `POST /api/v3/invoices`, send a unique `RequestId` request header. If the same `RequestId` is retried (after a timeout or dropped connection), the SDC returns the **same** already-signed invoice instead of signing again — the header is echoed back on the response. Always send a fresh `RequestId` per sale and reuse it only when retrying that exact sale.

Content type & CORS

- Requests and responses are JSON (`Content-Type: application/json`), except `POST /api/v3/pin`, whose body is the PIN as plain text.
- CORS is open for `GET`, `POST` and `OPTIONS` from any origin, so a browser-based POS can call the service directly.
- If the POS runs on another machine, allow inbound TCP **8888** through the Windows firewall on the SDC host.

Status & Session

These endpoints establish that the SDC is ready and unlock signing for the session.

GET /api/v3/attention

Lightweight health check. Returns whether the SDC is available and which card readers are connected.

```
GET /api/v3/attention

200 OK
{
  "available": true,
  "readers": ["ACS ACR39U ICC Reader 0"]
}
```

GET /api/v3/status

The E-SDC status document. Read this before signing.

Field	Meaning
isPinRequired	true if a PIN must be verified before signing.
auditRequired	true when the Secure Element has reached its limit and an audit is due.
sdcDateTime	The SDC's current date/time (use this as the fiscal clock).
protocolVersion	POS-to-SDC protocol version.
secureElementVersion	SE applet version.
make, model, softwareVersion, hardwareVersion, deviceSerialNumber	Device identity.
uid	Secure Element UID.
currentTaxRates	The tax-rate group currently in force (the labels/categories you may use on items).
allTaxRates	All known tax-rate groups (including future-dated).

Field	Meaning
<code>gsc</code>	Status codes, e.g. <code>1300</code> (SE not present), <code>2400</code> (SE limit reached).

POST /api/v3/pin

Verifies the operator PIN on the Secure Element and caches it in memory for the session. **The request body is the PIN as plain text** (digits) — not JSON. The response is a 4-digit text response code.

POST /api/v3/pin
Content-Type: text/plain
3456
200 OK
0100

Response	Meaning
<code>0100</code>	PIN verified — signing is unlocked.
<code>2100</code>	PIN incorrect / no usable PIN supplied.
<code>2110</code>	Blocked, or only one attempt remains (the SDC refuses the last attempt to avoid locking the card).
<code>1300</code>	Secure Element not present / unreadable.

Card safety: the SDC will not spend the final PIN attempt. If only one try remains it returns `2110` rather than risk permanently locking the card — resolve the correct PIN before retrying.

Secure Element helpers (read-only)

Useful for status displays and pre-flight checks; none of these sign or consume a PIN attempt.

Endpoint	Returns
<code>GET /api/v3/se/identity</code>	Seller identity from the SE certificate: <code>uid</code> , <code>tin</code> , <code>businessName</code> , <code>locationName</code> , <code>address</code> , <code>state</code> , <code>validFrom</code> , <code>validTo</code> , TaxCore URL.
<code>GET /api/v3/se/amount-status</code>	<code>saleRefund</code> (amount accumulated) and <code>limit</code> .
<code>GET /api/v3/se/pin-tries</code>	<code>triesLeft</code> .

Endpoint	Returns
GET /api/v3/se/safety	A safety snapshot: PIN tries left, whether blocked, amount used/limit/remaining, used fraction.
GET /api/v3/se/version	SE applet version.

Signing an Invoice

The core endpoint. Signs a sale on the Secure Element and returns the fiscal result (verification URL, QR, journal, signature). Requires a PIN previously verified via `POST /api/v3/pin`.

```
POST /api/v3/invoices
Content-Type: application/json
RequestId: 4ble...unique-per-sale
```

Request fields

Field	Type	Required	Notes
<code>invoiceType</code>	enum	Yes	<code>Normal</code> , <code>ProForma</code> , <code>Copy</code> , <code>Training</code> , <code>Advance</code> .
<code>transactionType</code>	enum	Yes	<code>Sale</code> or <code>Refund</code> .
<code>payment</code>	array	Yes	One or more <code>{ amount, paymentType }</code> . See payment types below.
<code>items</code>	array	Yes	One or more line items. See item fields below.
<code>dateAndTimeOfIssue</code>	string	No	ISO date/time; defaults to the SDC clock if omitted.
<code>cashier</code>	string	No	≤ 50 chars.
<code>buyerId</code>	string	No	≤ 20 chars (buyer TIN / ID).
<code>buyerCostCenterId</code>	string	No	≤ 50 chars.
<code>invoiceNumber</code>	string	No	≤ 60 chars (your POS's own number/ordinal).
<code>referentDocumentNumber</code>	string	Conditional	≤ 50 chars. The signed number of the original invoice — required for <code>Copy</code> and <code>Refund</code> , and whenever <code>referentDocumentDT</code> is present.
<code>referentDocumentDT</code>	string	No	Date/time of the referent document; if present, <code>referentDocumentNumber</code> becomes required.

Field	Type	Required	Notes
options	object	No	Key/value flags, e.g. { "omitTextualRepresentation": 1 } to skip building the journal text.

Item fields

Field	Type	Required	Notes
name	string	Yes	1-2048 chars.
quantity	number	Yes	≥ 0.001.
unitPrice	number	Yes	Price per unit.
labels	array	Yes	Tax label(s) for the item, e.g. ["A"]. Must be labels present in the active tax group (unknown labels are rejected).
totalAmount	number	Yes	Line total (quantity × unitPrice).
gtin	string	No	8-14 chars if supplied.

Payment types

Other, Cash, Card, Check, WireTransfer, Voucher, MobileMoney.

Example request

```
{
  "invoiceType": "Normal",
  "transactionType": "Sale",
  "cashier": "Jack",
  "payment": [
    { "amount": 23.00, "paymentType": "Cash" }
  ],
  "items": [
    {
      "name": "Coffee",
      "quantity": 2,
```

```

    "unitPrice": 5.00,
    "labels": ["A"],
    "totalAmount": 10.00
  },
  {
    "name": "Sandwich",
    "quantity": 1,
    "unitPrice": 13.00,
    "labels": ["A"],
    "totalAmount": 13.00
  }
]
}

```

Response

On success the SDC returns the fiscal result. Field names are camelCase (see Swagger for the exact schema of your build):

Field	Meaning
<code>invoiceNumber</code>	The signed fiscal invoice number.
<code>invoiceCounter</code> / <code>invoiceCounterExtension</code>	Per-type and combined counters.
<code>totalCounter</code> / <code>transactionTypeCounter</code>	Running counters from the SE.
<code>verificationUrl</code>	The URL a customer uses to verify the invoice with FRCS.
<code>verificationQRCode</code>	The QR code (base64) encoding the verification URL — print this on the receipt.
<code>journal</code>	The formatted textual receipt (40-column). Omitted if <code>options.omitTextualRepresentation</code> is set.
<code>signature</code> / <code>encryptedInternalData</code>	The SE signature and encrypted internal data.
<code>signedBy</code> / <code>requestedBy</code>	SE / requesting-signer identifiers.
<code>taxItems</code>	Per-category tax lines: <code>label</code> , <code>categoryName</code> , <code>rate</code> , <code>amount</code> .
<code>totalAmount</code>	Invoice total.
<code>taxGroupRevision</code>	Revision of the tax group applied.
<code>businessName</code> , <code>tin</code> , <code>locationName</code> , <code>address</code> , <code>district</code> , <code>mrc</code>	Seller details for the receipt header.
<code>sdcDateTime</code>	The signing date/time.

200 OK

```
{
  "invoiceNumber": "EKLBHT3Y-EKLBHT3Y-31",
  "verificationUrl": "https://sandbox.vms.fracs.org.fj/v/?vl=...",
  "verificationQRCode": "iVBORw0KGgoAAAANSUgAA...",
  "journal": "===== FISCAL INVOICE ===== ...",
  "totalAmount": 23.00,
  "taxItems": [
    { "label": "A", "categoryName": "VAT", "rate": 15.0, "amount": 3.00 }
  ],
  "sdcDateTime": "2026-08-13T09:42:00"
}
```

Refunds & copies

For a `Refund` (or a `Copy`), set `transactionType`/`invoiceType` accordingly and reference the original signed invoice with `referentDocumentNumber` (and optionally `referentDocumentDT`). Taxes for a copy/refund are calculated using the tax group in force at the referent document's date.

Validate without signing

`POST /api/v3/invoices/validate` runs the same request validation and tax rules **without** touching the card — useful to check a basket before committing. It returns `{ "valid": true }` or a 400 with field errors (see "Errors & Validation").

Errors, Response Codes & Validation

The SDC uses two error shapes: **field validation errors** (HTTP 400) and **operational response codes** (in the body/status).

Validation errors (HTTP 400)

When a request fails structural or business validation, the SDC returns HTTP 400 with a `modelState` array — one entry per invalid field, each carrying one or more 4-digit codes:

```
400 Bad Request
{
  "message": "Bad Request",
  "modelState": [
    { "property": "items[0].name", "errors": ["2803"] },
    { "property": "payment", "errors": ["2801"] }
  ]
}
```

Code	Meaning
2800	Field required.
2801	List length less than expected (e.g. no items / no payment).
2803	Field required (item field).
2804	Field length invalid.
2805	Field value invalid (bad type, unparseable date/number, unknown enum).
2807	Field out of range.

The `property` path mirrors the request, including array indices — e.g. `items[2].quantity`, `payment[0].paymentType`.

Operational response codes

These signal SDC/Secure-Element state rather than a malformed request.

Code	Where	Meaning
0100	POST /pin body	PIN verified.
1300	/pin body, /status.gsc	Secure Element not present / unreadable.
1500	/invoices 400 body	PIN verification required — call POST /api/v3/pin first.
2100	/pin body	PIN incorrect.
2110	/pin body	Blocked, or last attempt withheld to avoid locking the card.
2400	/status.gsc	Secure Element limit reached — audit required.

Missing PIN on sign

400 Bad Request

```
{ "message": "PIN verification required.", "code": "1500" }
```

Secure Element / card faults

A card fault during signing (removed card, PIN lock, SE limit) returns HTTP 503 with the message and the 4-digit code, for example:

503 Service Unavailable

```
"Secure Element limit reached (code 2210)"
```

Recommended handling

- **400 with `modelState`** — fix the offending field(s) and resubmit; do not retry unchanged.
- **1500** — prompt for the PIN, call `/pin`, then resubmit the sale (reuse the same `RequestId`).
- **503 (SE fault)** — surface the message to the operator; retry the same `RequestId` once the fault clears (e.g. card reinserted). Idempotency ensures no double-signing.

History & Reports

Read-back endpoints for reprinting, reconciliation and reporting. These query stored records and do not touch the card.

Invoice history

```
GET /api/v3/invoices?from=2026-08-01&to=2026-08-13&uid=EKLBHT3Y
```

Returns fiscalised invoice records for the period (defaults to the last 30 days). Each row includes `dateAndTime`, `invoiceNumber`, `uid`, `type`, `transaction`, `amount`, `referenceDocumentNumber`, `source` and `itemCount`. `uid` is optional.

Single invoice (drill-down)

```
GET /api/v3/invoices/detail?number=EKLBHT3Y-EKLBHT3Y-31
```

Returns the full stored record for one invoice: items, per-invoice tax breakdown and payments. `404` if the number is unknown.

Receipt PDF (reprint)

```
GET /api/v3/invoices/journal-pdf?number=EKLBHT3Y-EKLBHT3Y-31
```

Returns the rendered receipt as `application/pdf` — use it to reprint from history. `404` if the invoice has no stored textual representation (e.g. it was signed with `omitTextualRepresentation`).

Recovery by RequestId

```
GET /api/v3/invoices/{requestId}
```

Returns the invoice previously signed under that `RequestId` (or null). Useful after a lost response — no re-signing occurs.

Reports

All report endpoints take `from`, `to` and optional `uid`.

Endpoint	Returns
<code>GET /api/v3/reports/daily</code>	Daily totals.
<code>GET /api/v3/reports/payment-types</code>	Totals by payment type.
<code>GET /api/v3/reports/tax-amounts</code>	Tax totals by label/category.
<code>GET /api/v3/reports/list-of-items</code>	Item sales.
<code>GET /api/v3/reports/detailed</code>	A combined detailed report (summary + all breakdowns).
<code>GET /api/v3/reports/detailed/excel</code>	The detailed report as a multi-sheet <code>.xlsx</code> workbook.

Audit status

```
GET /api/v3/audit/status?uid=EKLBHT3Y
```

Returns `queuedPackages` (unsent audit packages), `awaitingProof` (a proof-of-audit is pending) and `lastAuditStartUtc`. Auditing is automatic in the background; this is for visibility, not something the POS drives.

Legacy V2 API

The Legacy V2 API exists for compatibility with older POS integrations (the FiscoBridge-era contract). New integrations should use v3. V2 signing reuses the exact same fiscalisation pipeline as v3 — only the request parser and the returned result shape differ.

Endpoints

Endpoint	Purpose
GET /api/attention	Availability + readers.
GET /api/status	Availability + readers.
POST /api/sign	Sign an invoice (V2 request → V2 result).
POST /api/sign/SignInvoice	Alias of /api/sign.

PIN verification uses the same POST /api/v3/pin as v3 — the cached PIN is shared across both APIs.

Key differences from v3

- **Single payment type.** The V2 request carries one request-level paymentType (not a payment array). A single payment is synthesised with an amount equal to the sum of item totals.
- **Stringly-typed nested fields.** items and options are provided as JSON strings, and item labels is a JSON string such as "[\"A\"]".
- **buyerCostCenterId** limit is 15 chars in V2 (50 in v3).
- **unitPrice** defaults to 1 if omitted (v3 requires it).
- **Result shape** is the abbreviated legacy FiscalizationResultV2 (short field names) rather than the full-named v3 InvoiceResult. The underlying values (verification URL, QR, journal, signature, counters) are the same.

Request fields (V2)

Field	Notes
invoiceType	Normal / ProForma / Copy / Training / Advance.
transactionType	Sale / Refund.

Field	Notes
paymentType	Single value: Cash, Card, etc.
items	JSON string of an array of { gtin, name, quantity, unitPrice, labels, totalAmount }; labels itself is a JSON string like "[\"A\"]".
dateAndTimeOfIssue, cashier, buyerId, buyerCostCenterId, invoiceNumber, referentDocumentNumber, options	As v3, with the V2 limits noted above.

Validation

V2 validation returns the same 400 {modelState} shape as v3; a parse failure (unparseable date, wrong JSON type) is reported as {2805} (field value invalid). You can dry-run a V2 request with {POST /api/v3/invoices/v2/validate}.

Recommendation

Use V2 only to keep an existing POS working. For new development, target v3 for the typed {payment} array, the richer result fields, and {RequestId} idempotency.