Protocols

This section describes Application Programming Interfaces (API) and protocols exposed by an E-SDC or used by an E-SDC to communicate with the other components (TaxCore.API, Secure element Applet, PKI Applet or SD Card/USB Flash Drive) required to fulfill its primary role – to safeguard a transaction and to transfer the audit packages to the tax authority's system.

Accredited POS systems can communicate with the E-SDC using the POS to SDC Protocol.

1. TaxCore.API for E SDC
   TaxCore.API for E-SDCs is a REST API exposed by a tax authority's system to E-SDC devices. It provides services used by the E-SDCs to submit Audit Packages, to notify TaxCore if the online status has been changed, and to receive configuration commands.

2. Secure Element Applet API
   Communication with a Secure element Applet API is performed through standard APDU commands.

3. File Based Communication
   This section contains the description of the **File-based** communication with E-SDC.

TaxCore.API for E-SDC

TaxCore.API for E-SDCs is a REST API exposed by a tax authority's system to E-SDC devices. It provides services used by the E-SDCs to submit Audit Packages, to notify TaxCore if the online status has been changed, and to receive configuration commands.

An E-SDC is authenticated by TaxCore.API using a client digital certificate and an authentication token.

NOTE:
For more information about TaxCore.API that applies to **both** E-SDCs and POS's, refer to the documents in this section.

Read more

1. Authentication

Communication between a Client and TaxCore.API is carried out via the HTTPS protocol.

2. SDC
   The SDC Section of TaxCore.API is used by any SDCs to establish a connection with backend.

Authentication

Introduction

Communication between a Client and TaxCore.API is carried out via the HTTPS protocol.

The Client is authenticated by TaxCore.API using either a client certificate or an authentication token obtained from TaxCore.API. To obtain an authentication token, a client certificate authentication has to be successfully conducted as the first step. For more information, see Request Authentication Token.

Once the token has been obtained, the HTTP request must contain *TaxCoreAuthenticationToken* key in the request HTTP headers with a valid authentication token as a value.

Digital Certificates and PIN Codes

The tax authority's system issues a Secure Element to a taxpayer as follows:

1. Taxpayer's digital certificate is stored in the Secure Element;
2. The Secure Element is stored on the smart card;
3. The PIN or password is generated and printed on the PIN mailer;
4. The Secure Element and PIN code are securely delivered to the taxpayer.

Digital Certificates for Testing Purposes

The tax authority will issue the requested number of test digital certificates to each accredited supplier and each accredited taxpayer.

Authentication Token

The E-SDC uses an authentication token when calling the TaxCore API web services. The authentication token is obtained from TaxCore.API by calling the service Request Authentication Token and providing a Taxpayer's digital

certificate.

Role of the PKI Applet

PKI (public key infrastructure) Applet is installed along with the Secure Element Applet on the same smart card.

The role of the PKI Applet is to support secure communication and client certificate authentication with TaxCore.API using HTTPS protocol. The certificate used to establish a secure connection is stored on a smart card and it can be accessed from the PKI Applet using PKSC#11 API.

The certificate is loaded in the slot / token structure on the PKI Applet.

After the certificate is extracted from the smart card (in DER format), it can be used as a standard X.509 certificate for TLS/SSL and HTTPS protocols.

A valid PIN is required to read the certificate from the PKI Applet using PKCS#11 API. The PIN for PKI Applet is the same as the PIN for the Secure Element Applet.

Content

Required Drivers

Introduction

Smart cards are programmed with PKI firmware according to the GIDS (Generic Identity Device Specification) standard. Appropriate drivers will be installed/programmed on an E-SDC in order to enable PKI Applet usage.

Windows OS Drivers

GIDS driver is an integral part of Windows OS since Windows 7 SP1, enabling the instant use of a smart card. No additional driver installation is required.

Linux OS Drivers

In order to use PKI Applet on a Linux based OS, a pkcs11 driver from the OpenSC library is required. OpenSC libraries and tools are freely available on https://github.com/OpenSC.

In the following example, the installation of required drivers, libraries, and tools on Debian / Ubuntu flavor of Linux OS with USB based card reader is shown. It is assumed that OpenSSL is used for TLS/SSL communication.

1. Install card reader driver

```
apt-get install libudev-dev
wget https://alioth.debian.org/frs/download.php/file/4126/pcsc-lite-x.y.z.tar.bz2
tar -xf pcsc-lite-x.y.z.tar.bz2
cd pcsc-lite-x.y.z
./configure
make
make install
aptitude install libusb-1.0-0-dev
wget https://alioth.debian.org/frs/download.php/file/4111/ccid-x.y.z.tar.bz2
tar -xf ccid-x.y.z.tar.bz2
cd ccid-x.y.z
./configure
make
make install
copy 92_pcscd_ccid.rules file from src directory to /etc/udev/rules.d/
# aptitude install libltdl-dev
wget http://ftp.de.debian.org/debian/pool/main/o/openct/openct_x.y.z.orig.tar.gz
tar -xf openct_x.y.z.orig.tar.gz
cd openct_x.y.z
./configure
# make
make all
```

2. Install OpenSSL development library

```
apt-get install libssl-dev
```

3. Install OpenSC package

```
wget http://cznic.dl.sourceforge.net/project/opensc/OpenSC/opensc-x.y.z/opensc-x.y.z.tar.gz
tar -xf opensc-x.y.z.tar.gz
cd opensc-x.y.z
./configure
make
make install
```

Run the opensc-tool command from the terminal

If you get the message that libopensc.so.3 cannot be loaded, find it with

```
find / -name "libopensc.so"
```

Copy found library to /usr/lib

4.

Install libp11 library

```
apt-get install libp11-2
```

5. Install engine_pkcs11 library

- Download source code from https://github.com/OpenSC/engine_pkcs11/releases/
- Build and install library according to instructions found project page

After the above steps are executed, the certificate will be accessible from the appropriate slot/token using a PKCS11 family of functions from the lipb11 library. ENGINE family of functions can be used to load the pkcs11 engine in the OpenSSL.

Other Platforms and Operating Systems

Please contact OpenSC community (https://github.com/OpenSC) for further information.

SDC

Introduction

The SDC Section of TaxCore.API is used by any SDCs to establish a connection with backend.

Activities

- Get initialization and configuration information required for normal operation;
- Submit prepared audit packages;
- Notify backend of online/offline status;
- Pull pending commands from backend;
- Notify backend of command execution results.

Content

1.

Request Authentication Token
When requesting the authentication token, a Client authenticates themselves with a valid Digital Certificate (stored in the PKI applet). If the token is successfully created, it is returned to the Client as a string. In order to receive an authentication token, each Client must establish a secure connection to the "/api/v3/sdc/token" endpoint on TaxCore.API and authenticate using client digital certificate.

2. Get Initialization Commands
For each new smart card issued by a tax authority, a set of commands is generated, which contain information necessary for invoice signing (Tax Rates, Verification URL, NTP server, etc.).

3. Notify Online Status
If an E-SDC is online, it will periodically (once every 1 – 5 minutes) invoke the "Notify Online Status" operation on TaxCore.API.

4. Notify Command Processed
After an E-SDC processes commands received from TaxCore.API, it will report the results of execution to TaxCore.API.

5. Submit Audit Package
After the invoice audit package is created (explained in the section Creating an Audit Package), it will be transferred to TaxCore.API the next time an Internet connection is available.

6. Submit Audit Request Payload ARP
The E-SDC invokes the Start Audit APDU command and receives 260 bytes of data that represent the Audit Request Payload (ARP). ARP has to be converted to the string using Base64 encoding.

Request Authentication Token

Introduction

When requesting the authentication token, a Client authenticates themselves with a valid Digital Certificate (stored in the PKI applet). If the token is successfully created, it is returned to the Client as a string. In order to receive an authentication token, each Client must establish a secure connection to the "/api/v3/sdc/token" endpoint on TaxCore.API and authenticate using client digital certificate.

A request is composed as follows:

1. Create a HTTPS GET request object;
2. Add HTTP headers "Accept: application/json" and "Content-Type: application/json";
3. Read the certificate from the PKI Applet;
4.

Use the certificate from the PKI Applet to establish a SSL/TLS connection;
5. Send a request to the "/api/v3/sdc/token" operation on the TaxCore.API web service;
6. Read the response as a JSON structure defined below.

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/token	`[[_TaxCore.PublicConfiguration.TaxCoreApi`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code, and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

`GET`

Header

Add the following HTTP headers to each request:

- `Accept: application/json`

Authentication

The certificate-based authentication is used only to request a token. Request for the authentication token is periodically invoked to obtain a new token and to verify the date and time.

For authentication details please refer to Authentication

Request

N/A

Response

Field	Type	Description
token	string	The token is valid for 8 hours by default. A Client uses the current token when calling all other services exposed by TaxCore.API.
expiresAt	string	Date and time of token expiration - in UTC time. When a token expires, a Client must request a new token. If the Client requests a new token while the current token is still valid, TaxCore will return the current token.

Example

```
{
  "token": "245ebd69-1438-4dc3-a65b-18f1a527f093",
  "expiresAt": "2020-12-23 15:18:33Z"
}
```

Get Initialization Commands

Introduction

For each new smart card issued by a tax authority, a set of commands is generated, which contain information necessary for invoice signing (Tax Rates, Verification URL, NTP server, etc.).

Commands can be downloaded using one of the following channels:

- By invoking TaxCore.API service Notify Online Status (typically by E-SDC);
- By invoking TaxCore.API service Submit Audit Package (typically by E-SDC);
- By invoking TaxCore.API service Get Initialization Commands (typically by E-SDC); or
- By using the Taxpayer Administration Portal.

Once the commands are processed, the E-SDC reports the execution status to TaxCore.API as explained in the section Notify Command Processed.

The E-SDC can explicitly require initialization commands, by invoking the TaxCore.API service *Get Initialization Commands*.

Initialization commands include:

- *Configure Time Server URL Command*;
- *Set Tax Rates Command*;
- *Update Verification URL Command*;
- *Update TaxCore Configuration*.

To get initialization commands, compose a HTTPS GET request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json", and header that contains an authentication token
2. Submit GET request to https://<taxcore_api_url>/api/v3/sdc/commands.

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e here>/api/v3/sdc/commands	`[[_TaxCore.PublicConfiguration.TaxCoreApi`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code, and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

`GET`

Header

Add the following HTTP headers to each request:

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`

Authentication

For authentication details please refer to Authentication

Request

N/A

Response

The response contains a list of commands that must be executed by E-SDC, as described in section Commands.

Notify Online Status

Introduction

If an E-SDC is online, it will periodically (once every 1 – 5 minutes) invoke the “Notify Online Status” operation on TaxCore.API.

Compose a HTTPS PUT request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json" and a header that contains the authentication token;
2. Add a string “true” or "false" to the body of the request (depending on whether the E-SDC is online or going offline);
3. Submit a PUT request to the https://<taxcore_api_url>/api/v3/sdc/status.

After the request is sent, TaxCore.API will return a response with a JSON formatted string containing a list of commands, as described in the section Commands, which an E-SDC will execute (new tax rates, verification URL, NTP URL, or public key used for encryption). The Command list can be empty.

Endpoints

Endpoint	Example
`[[_TaxCore.PublicConfiguration.TaxCoreApiUrl`	`https://api.sandbox.taxcore.online/a`

Method

`PUT`

Header

Add the following HTTP headers to each request:

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details, please refer to Authentication.

Request

true

The list of commands will be obtained only if the string "true" is submitted in the request.

Response

The response contains a list of commands that must be executed by the E-SDC, as described in the section Commands.

Notify Command Processed

Introduction

After an E-SDC processes commands received from TaxCore.API, it will report the results of execution to TaxCore.API.

1. Add headers "Accept: application/json", "Content-Type: application/json", and a header that contains the authentication token;
2. Add a string “true” or "false" to the body of the request (depending on whether the E-SDC successfully processed commands);
3. Submit a PUT request to https://<taxcore_api_url>/api/v3/sdc/commands/{commandId}.

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/commands/{commandId}	`[[_TaxCore.PublicConfiguration.TaxCoreApi 205A-4CBF-AD0C-6617D42AE466`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code, and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

`PUT`

Header

Add the following HTTP headers to each request:

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details please refer to Authentication.

Request

true

Example

https://api.sandbox.taxcore.online/api/v3/sdc/commands/CC63C53D-205A-4CBF-AD0C-6617D42AE466

Response

HTTP 200 OK

Submit Audit Package

After the invoice audit package is created (explained in the section Creating an Audit Package), it will be transferred to TaxCore.API the next time an Internet connection is available.

Compose a HTTPS POST request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json", and a header that contains an authentication token
2. Add an Audit Package as a JSON message to the body of the HTTP POST request;
3. Submit a POST request to https://<taxcore_api_url>/api/v3/sdc/audit.

After the request is sent, TaxCore.API responds with a JSON formatted text containing a status of operation and a list of commands that an E-SDC will execute.

NOTE:
Values for `sdcDateTime` and all other fields (see Create Invoice) must match the values submitted to the Secure Element for digital signing (see *Sign Invoice* in Fiscalization), as well as the values in the verification URL (see Create Verification URL).

NOTE:
In case the field `Request.Item.Name` exceeds the defined maximum length (see Create Invoice), its value is truncated.

NOTE:
In case the field `Request.DateAndTimeOfIssue` is out of range (see Create Invoice), its value is replaced with a null value.

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/audit	`[[_TaxCore.PublicConfiguration.TaxCoreApi`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code, and UI should not be hard-coded but configurable or extracted from a digital certificate.

Method

`POST`

Header

Add the following HTTP headers to each request:

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details, please refer to Authentication.

Request

The request that must be generated and sent by the SDC is described in this section Format of the Audit Package.

Example

```
{
  "key": "VGhpcyBJcyBLZXkK...",
  "iv": "VGhpcyBJcyBJVgo...",
  "payload": "VGhpcyBJcyBQYXlsb2FkCg..."
}
```

Response

The response contains a list of commands that must be executed by the E-SDC, as described in the section Commands.

```
AuditDataStatus {
status (integer, optional) = ['0', '1', '2', '3', '4', '5', '6']integerEnum:0, 1, 2, 3, 4, 5,
commands (Array[Command], optional)
}]
```

Data Fields

status - returned after TaxCore.API unpacks and verifies audit packages. If all verifications are successful, the status should have the value **4**. Other values can help E-SDC developers rectify problems with audit packages. Their meanings are the following:

- **0** - Invalid audit package - TaxCore API failed to decrypt the received audit package. The possible reasons are: the received file is corrupt, has no content, the file is encrypted using the wrong TaxCore public key, etc.
- **1** - Invoice cannot be stored - the received invoice is probably valid, but due to the internal server error TaxCore.API is unable to store it
- **2** - Signature is invalid
- **3** - Invoice internal data was encrypted in a wrong way
- **4** - Invoice is verified
- **5** - Taxes on this invoice were calculated using a tax rates group that does not exist or is obsolete

- **6** - At the moment of invoice signing, the certificate on the secure element was already revoked
- **8** - At the moment of invoice signing, the certificate on the secure element was not officially issued
- **24** - Information about the E-SDC's Manufacturer Registration Code is in a wrong format
- **69** - Invoice contains TaxItem for a label that does not exist in the tax rate group named in `Result.TaxGroupRevision`
- **70** - Invoice contains TaxCounters (in internal data) for CategoryOrderId which does not exist in the tax rate group named in `Result.TaxGroupRevision`, i.e. E-SDC sent a non-existing CategoryOrderId to the secure element
- **72** - E-SDC sent the wrong TIN or SignedBy or RequestedBy fields when signing this invoice
- **commands** - contains a list of commands that E-SDC should execute, as described in section Commands.

NOTE:
If status 4 (*Invoice is verified*) is received from TaxCore.API, that audit package should immediately be deleted from the E-SDC's local storage (see Creating an Audit Package). If any other status is received, the audit package must not be deleted.

NOTE:
The E-SDC should **try to resubmit** an audit package to TaxCore.API. **only** if it receives status 1 (*Invoice cannot be stored*) from the TaxCore.API. If any other status is received, the audit package should not be resubmitted.

Example

```
{
  "status": 0,
  "commands": [
    {
      "commandId": "3930CEEF-F637-444D-8295-F629D6E482D3",
      "type": 1,
      "payload": "0.europe.pool.ntp.org",
      "uid": "ABCD1234"
    }
  ]
}
```

Submit Audit Request Payload - ARP

The E-SDC invokes the Start Audit APDU command and receives 260 bytes of data that represent the Audit Request Payload (ARP). ARP has to be converted to the string using Base64 encoding.

The E-SDC invokes the Amount Status APDU command and receives the current sum and limit for the secure element.

These 3 values are submitted to the endpoint https://<taxcore_api_url>/api/v3/sdc/audit-proof as an Audit-Proof Request structure in the body of the HTTP request.

Compose a HTTPS POST request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json", and a header that contains the authentication token;
2. Create a request structure as per the below model and add it to the body of the HTTP POST request;
3. Submit a POST Request to https://<taxcore_api_url>/api/v3/sdc/audit-proof.

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/audit-proof	`[[_TaxCore.PublicConfiguration.TaxCoreApi proof`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code, and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

`POST`

Header

Add the following HTTP headers to each request:

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`

- Accept: application/json
- Content-Type: application/json

Authentication

For authentication details please refer to Authentication

Request

JSON structure as defined in section Format of the Audit-Proof Request

Model

```
ProofOfAuditRequest {
auditRequestPayload (string),

sum (integer) 64bit unsigned,

limit (integer) 64bit unsigned
}
```

Example

```
{
  "auditRequestPayload": "d4A/iLtwmDYeZyacm/nDlCF...",
  "sum": 11034,
  "limit": 100000
}
```

Response

```
HTTP 200 OK
```

Secure Element Applet API

Communication with a Secure element Applet API is performed through standard APDU commands.

For a detailed description of APDU communication, APDU commands data structure, and particular bytes meaning, please refer to ISO/IEC 7816-4 standard.

Commands are grouped into three categories based on the type of usage:

1. General
2. Fiscalization
3. Audit

Important Notes

1. From **SE applet version 2.0.0**
    1. All APDU commands are sent to the Smart Card using T1 communication protocol
    2. All amounts or counter values are submitted to/received from the Secure element using Big-endian. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address)
    3. P1 and P2 values are considered in the request processing when,
        1. Select Applet Command
        2. Force using CRC for Data in APDU transmission
2. From **SE applet version 3.1.1**, Get Last Signed Invoice was introduced, replay of last signed invoce data
3. From **SE applet version 3.2.2**, PIN improvements were introduced
    1. PIN is sent in the ASCII hex format
    2. Same PIN object (*value*) is used for SE and PKI applet,
4. From **SE applet version 3.2.5**, ISO/IEC 3309 compliant 32-bit CRC (Cyclic Redundancy Check) algorithm - **CRC-32/ISO-HDLC** - is avaliable, and it is optional to use,
    1. Algorithm:
        1. Initialize with 0xFFFFFFFF,
        2. The input data is reversed (reflected),
        3. The ISO 3309 algorithm is used with the polynomial value 0x04C11DB7,
        4. The resulting 32 bit FCS is reversed (reflected),
        5. The reversed 32 bit FCS is xor'd with OxFFFFFFFF,
    2. Example: for input `0x01020304`, it will produce CRC-32 `0xB63CFBCD`,
    3. P1 and P2 values are used to force CRC for Data in APDU transmission,
5. From **SE applet version 3.2.8**, get CertParams was introduced, to get UID and Certificate NotBefore and NotAfter,
6. From **SE applet version 3.2.9**, PIN can be sent in both ASCII and decimal hex formats as backward compatibility. ASCII hex format is considered the default behavior,
7. From **SE applet version 3.2.10**, if the SDC Time on an invoice is outside of the secure element's certificate validity period (i.e., before the NotBefore limit or after the NotAfter limit), the secure element returns error code 0x6308,
8. From **SE applet version 3.2.12**, the enforcement to check whether the SDC Time on an invoice is outside the secure element's certificate validity period (i.e., before the NotBefore limit or after the NotAfter limit) can be disabled/enabled.

Content

Fiscalization
PIN verification is a method that “unlocks” a card for invoice signing and other operations protected by PIN code. Depending on the SE applet version, PIN is sent in decimal or hex format with ASCII encoding, and it is sent as an array of byte digits.

3. Audit
Returns 259 bytes data structure represents public card key (256 bytes modulus and 3 bytes exponent). This key is used to encrypt Audit packages.

4. Secure Element Specific APDU Error Codes
This table contains the expected error codes and descriptions that a caller may encounter while working with the Secure Element Applet.

General Commands

Secure Element Applet is installed as a non-default applet on a smart card. Before any APDU command is invoked, the applet is selected using the standard Select command.

NOTE:
The availability of specific commands, as well as their content, depends on the secure element (SE) version. You can use the *Get Secure Element Version* command (see below) to check the version of the SE you are using.

Select SE Applet

As previously mentioned, a Smart Card has two applets installed. This command selects the Secure Element Applet and routes subsequent APDU commands to it.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expecte Lengtl (Le)
>= 2.0.0	Case3Sho	0x00	0xA4	0x040(	0x10	0xA000000748	0x00

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	none	0x9000

Example:

Request: 00A4040010A000000748464A492D546178436F726500

Response: 9000

Get Secure Element Version

This command returns the version information about the current API version. The response contains 12 bytes, where each 4 bytes represent unsigned integer of one version segment, making total of 3 version segments: major, minor, and patch.

APDU Request

SE CAP Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	Case2Sho	0x88	0x08	0x000(	none	none	0x00

APDU Response

SE CAP Version	Response Data	SW1SW2
>= 2.0.0	*12 bytes*	0x9000

Example 1:

Request: 8808040000

Response: 000000020000000000000000 9000

Example 2:

Request: 8808000000

Response: `000000030000000100000001 9000`

Example 3:

Request: `8808000000`

Response: `000000030000000200000005 9000`

Forward Secure Element Directive

This command is used by E-SDC to forward instructions received from TaxCore.Api to Secure Element Applet via Secure Element APDU Command.

If APDU Command status (SW1SW2) is OK (`0x9000`), consider that forward instructions operation is completed.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case3Ext`	`0x88`	`0x40`	`0x040(`	`0x000200`	*512 bytes received from TaxCore*	none
>= 3.2.5 (with CRC)	`Case3Ext`	`0x88`	`0x40`	`0x0102`	`0x000204`	*512 bytes received from TaxCore + 4 bytes for CRC*	none

APDU Response

SE Version	Response Data	SW1SW2

>= 2.0.0 (no CRC)	none	0x9000
>= 3.2.5 (with CRC)	none	0x9000

Example 1 (without CRC):

Command Data:
5DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E264255838C1DBAD

Request:
884004000002005DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E2

Response: 9000

Example 2 (with CRC):

Command Data witout CRC:
5DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E264255838C1DBAD

Command Data CRC: F50CFF4B

Command Data:
5DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E264255838C1DBAD

Request:
884004000002045DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E2

Response: 9000

Export Certificate

This command exports the taxpayer certificate in a DER format. This certificate contains location data that is present on the textual representation of an invoice.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	Case2Ext	0x88	0x04	0x040(	none	none	0x000000

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	*raw bytes random length*	0x9000

Example:

Request: 80404000000000

Response: *raw bytes of x509 certificate public key* + 9000

Get Last Signed Invoice

This command returns information about the last singed invoice. The structure of the data received is the same as the response in the Sign Invoice command is.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 3.1.1 (no CRC)	Case2Ext	0x88	0x15	0x040(	none	none	0x000000
>= 3.2.5 (with CRC)	Case2Ext	0x88	0x15	0x0102	none	none	0x000000

APDU Response

SE Version	Response Data	SW1SW2
>= 3.1.1 (no CRC)	*577 or 833 bytes*	0x9000
>= 3.2.5 (with CRC)	*581 or 837 bytes*	0x9000

Example 1 (without CRC):

Request: `88150400000000`

Response: *577 or 833 bytes* + `9000`

Response Data

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	Unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	

Example 2 (with CRC):

Request: `8815010200`

Response: *581 or 837* + `9000`

Response Data

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE

8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	Unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	
577 or 833	4	CRC	CRC is calculated from 0 to 577 or 833 bytes.

Get PIN tries left from SE Applet

This command returns how many PIN tries are left before the card is locked.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 3.1.1	`Case2Sho`	`0x00`	`0x16`	`0x040(`	none	none	`0x00`

APDU Response

SE Version	Response Data	SW1SW2

>= 3.1.1	05 if 5 tries are left, 00 if the card is blocked	0x9000

Example:

Request: 8816040000

Response: 05 9000

Get CertParams

This command returns UID, SE Certificate NotBefore, and SE Certificate NotAfter. NotBefore and NotAfter are in UTC time in Unix Timestamp format.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 3.2.8	Case2Sho	0x00	0x33	0x0000	none	none	0x00

APDU Response

SE Version	Response Data	SW1SW2
>= 3.2.8	*24 bytes*	0x9000

Example:

Request: 8833000000

Response: 445337584C53524500000196 8743CA28000001AC9386D1E8 9000

CertParam	Hex	Transform Value
UID	445337584C535245	DS7XLSRE
NotBefore	000001968743CA28	04/30/2025 15:14:49
NotAfter	000001AC9386D1E8	04/30/2028 15:24:49

Select PKI Applet

As previously mentioned, the Smart Card has two applets installed. This command selects the PKI applet and routes subsequent APDU commands to it.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
< 3.1.1	Case3Sho	0x00	0xA4	0x040(	0x0B	0xA00000	none
>= 3.1.1	Case3Sho	0x00	0xA4	0x040(	0x0B	0xA00000	none

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	none	0x9000

Example:

Request: 00A404000BA00000039742544659020 1

Response: 9000

Example:

Request: 00A404000BA00000039742544659000 9

Response: 9000

Fiscalization

PIN Verify

PIN verification is a method that “unlocks” a card for invoice signing and other operations protected by PIN code. Depending on the SE applet version, PIN is sent in decimal or hex format with ASCII encoding, and it is sent as an

array of byte digits.

For example, PIN 1234 can be represented in the following formats:

- decimal format - PIN is represented as 0x01, 0x02, 0x03, 0x04.
- ASCII hex format - PIN is represented as 0x31, 0x32, 0x33, 0x34.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
2.0.0 □ SE version < 3.2.2 3.2.9 □ SE version	`Case3Sho`	`0x88`	`0x11`	`0x000(`	`0x04`	*4 bytes where each represents one PIN digit in decimal format*	none
3.2.2 □ SE version	`Case3Sho`	`0x88`	`0x11`	`0x000(`	`0x04`	*4 bytes where each represents one PIN digit in ASCII hex format*	none

Example:

This is an example for PIN 1234.

SE Version	Command Data	Request	Response (correct PIN)	Error response (wrong PIN)
2.0.0 □ SE version < 3.2.2	`01020304`	`88110000040102030`	`9000`	`6302`
>= 3.2.2	`31323334`	`88110000043132333`	`9000`	`6302`

Sign Invoice

Signs invoice and returns fiscalization data for a submitted invoice.

NOTE:
From the SE version 3.2.5: Optional - CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

From applet version 3.2.8: Mandatory - Invoice Date/time must be greater then Certificate NotBefore and lower then Certificate NotAfter.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case4Ext`	`0x88`	`0x13`	`0x040(`	*3 byte Command Data byte array length*	*Command Data byte array*	`0x0000`
>= 3.2.5 (with CRC)	`Case4Ext`	`0x88`	`0x13`	`0x0102`	*3 byte Command Data byte array length*	*Command Data byte array + 4 bytes for CRC*	`0x0000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	*byte array*	`0x9000`
>= 3.2.5 (with CRC)	*byte array + 4 byte CRC*	`0x9000`

Data structure without CRC:

Command data:

Start (byte)	Length (byte)	Field	Description

0	8	Date/time	E-SDC timestamp UTC time in Unix Timestamp. Example: 1495018011910 is 2017-05-17T10:46:51.910Z
8	20	Taxpayer ID	Hex encoded byte array, leading bytes filled with 0x00. Taxpayer ID value can consist only of ascii printable characters. **Zeros can be added only on the left side**. MSB are sent first Example: Taxpayer ID = 928615467, Byte array = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x39, 0x32, 0x38, 0x36, 0x31, 0x35, 0x34, 0x36, 0x37} (byte 0x37 is sent last to SE)
28	20	Buyer ID	If unknown, leave zeroes. Formatting is the same as for Taxpayer ID
48	1	Invoice type	Values 0, 1, 2, 3, 4 as explained in section Create Invoice.
49	1	Transaction Type	Sale=0, Refund=1
50	7	Invoice amount	Sale or refund total amount (including taxes) - depends on applied tax types
57	1	Number of tax categories	Defines the number of tax categories which appear on the invoice (value between 0 and 12). The following data structure **Tax Categories** must be repeated exactly this number of times.
58	8	Tax Category (1)	The first Tax Category (mandatory if **Number of tax categories > 0**)
66	8	Tax Category (2)	The second Tax Category (mandatory if **Number of tax categories > 1**)
74	...	Tax Category (n)	

Tax Categories:

Start (byte)	Length (byte)	Field	Description
58	[1]	[Tax category ID]	The first tax category's OrderID, as explained in Tax Rates section (mandatory if **Number of tax categories** > 0)
59	[7]	[Tax category amount]	The first total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of**

			tax categories > 0)
66	[1]	[Tax category ID]	The next tax category's OrderID (mandatory if **Number of tax categories** > 1)
67	[7]	[Tax category amount]	The next total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 1)

Response data:

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	

Example without CRC:

Command Data:
`0000017B2D198AC40000000000000000050432D313030303030303031000000000000000000000000000000`

Request:
`8813040000009A0000017BE9B01AB40000000000000000050432D313030303030303031000000000000000`

Response: *byte array* + `9000`

Data structure with CRC:

Command data:

Start (byte)	Length (byte)	Field	Description
0	8	Date/time	E-SDC timestamp UTC time in Unix Timestamp. Example: 1495018011910 is 2017-05-17T10:46:51.910Z
8	20	Taxpayer ID	Hex encoded byte array, leading bytes filled with 0x00. Taxpayer ID value can consist only of ascii printable characters. **Zeros can be added only on the left side**. MSB are sent first Example: Taxpayer ID = 928615467, Byte array = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x39, 0x32, 0x38, 0x36, 0x31, 0x35, 0x34, 0x36, 0x37} (byte 0x37 is sent last to SE)
28	20	Buyer ID	If unknown, leave zeroes. Formatting is the same as for Taxpayer ID
48	1	Invoice type	Values 0, 1, 2, 3, 4 as explained in section Create Invoice.
49	1	Transaction Type	Sale=0, Refund=1
50	7	Invoice amount	Sale or refund total amount (including taxes) - depends on applied tax types
57	1	Number of tax categories	Defines the number of tax categories which appear on the invoice (value between 0 and 12). The following data structure **Tax Categories** must be repeated exactly this number of times.
58	8	Tax Category (1)	The first Tax Category (mandatory if **Number of tax categories > 0**)
66	8	Tax Category (2)	The second Tax Category (mandatory if **Number of tax categories > 1**)
74	...	Tax Category (n)	
...	4	CRC	CRC is calculated from 0 to 74 bytes (or to last byte if

			data).

Tax Categories:

Start (byte)	Length (byte)	Field	Description
58	[1]	[Tax category ID]	The first tax category's OrderID, as explained in Tax Rates section (mandatory if **Number of tax categories** > 0)
59	[7]	[Tax category amount]	The first total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 0)
66	[1]	[Tax category ID]	The next tax category's OrderID (mandatory if **Number of tax categories** > 1)
67	[7]	[Tax category amount]	The next total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 1)

Response data:

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed

			during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	
577 or 833	4	CRC	CRC is calculated from 0 to 577 or 833 bytes.

Example with CRC:

Command Data:
`0000017B2D198AC40000000000000000050432D313030303030303100000000000000000000000000000`

Command Data CRC: `90F2BC39`

Request:
`88130102E0000017BE9B01AB40000000000000000050432D313030303030303100000000000000000000`

Response: *byte array invoice + 4 byte CRC +* `9000`

Amount Status

Returns 14-bytes-long data structure (7 bytes for sum SALE and REFUND, and 7 bytes for Limit Amount)

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	`Case2Sho`	`0x88`	`0x14`	`0x040(`	none	none	`0x00`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	*14 byte array*	`0x9000`

Example:

Request: `8814040000`

Response: `0000724AA18328038D7EA4C68000 9000` (SALE+REFUND=490878370600 , Limit Amount=1000000000000000)

Audit

Export TaxCore Public Key

Returns 259 bytes data structure represents public card key (256 bytes modulus and 3 bytes exponent). This key is used to encrypt Audit packages.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	`Case2Ext`	`0x88`	`0x07`	`0x040(`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	*259 bytes data*	`0x0900`

Example:

Request: `88070400000000`

Response: *256 bytes modulus + 3 bytes exponent* + `9000`

Export Audit Data

Exports encrypted audit data.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	Case2Ext	0x88	0x12	0x040(	none	none	0x000000
>= 3.2.5 (with CRC)	Case2Ext	0x88	0x12	0x0102	none	none	0x000000

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	*565 or 821 bytes data*	0x9000
>= 3.2.5 (with CRC)	*569 or 825 bytes data*	0x9000

NOTE:
Depending on the Internal Data, the total length of the structure is 565 or 821 bytes. For versions **3.2.5 or later** if CRC is used, the total lenght can be 569 or 825 if CRC is added.

Exported audit data has the following structure, without CRC:

Offset	Length	Data	Note
0	4	TaxCore Key Version	
4	256	Crypted Internal Data	The length of Crypted Internal Data can be 256 or 512 bytes
260 or 516	20	Taxpayer Identification Number (TIN)	
280 or 536	20	Buyer ID	
300 or 556	1	Invoice type	
301 or 557	1	Transaction type	

302 or 558	7	Invoice amount	
309 or 565	256	Digital signature of the above structure	

Exported audit data has the following structure, with CRC:

Offset	Length	Data	Note
0	4	TaxCore Key Version	
4	256	Crypted Internal Data	The length of Crypted Internal Data can be 256 or 512 bytes
260 or 516	20	Taxpayer Identification Number (TIN)	
280 or 536	20	Buyer ID	
300 or 556	1	Invoice type	
301 or 557	1	Transaction type	
302 or 558	7	Invoice amount	
309 or 565	256	Digital signature of the above structure	
565 or 821	4	CRC	CRC is calculated from 0 to 565 or 821 bytes.

Example 1 (without CRC):

Request: `88120400000000`

Response: *565 or 821 bytes* + `9000`

Example 2 (with CRC):

Request: `88120102000000`

Response: *569 or 825 bytes* + `9000`

Start Audit

Notifies the Secure element that the audit process has been initialized by E-SDC.

Secure element returns an encrypted message that shall be submitted to TaxCore as the content of the field `auditRequestPayload` of audit-proof request.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case2Ext`	`0x88`	`0x21`	`0x040`	none	none	`0x000000`
>= 3.2.5 (with CRC)	`Case2Ext`	`0x88`	`0x21`	`0x010`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	*260 bytes data*	`0x9000`
>= 3.2.5 (with CRC)	*264 bytes data*	`0x9000`

Example 1 (without CRC):

Request: `88210400000000`

Response: *260 bytes data* + `9000`

Example 2 (with CRC):

Request: `88210102000000`

Response: *260 bytes data + 4 bytes CRC data* + `9000`

End Audit

Notifies the Secure element that the audit process has been finalized by TaxCore. If APDU Command status is OK (0x90 0x00) consider the audit operation is completed.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	Case3Ext	0x88	0x20	0x040(	0x000100	*256 bytes received from TaxCore*	none
>= 3.2.5 (with CRC)	Case3Ext	0x88	0x20	0x0102	0x000104	*256 bytes received from TaxCore + 4 bytes for CRC*	none

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	none	0x9000
>= 3.2.5 (with CRC)	none	0x9000

Example 1 (without CRC):

Command Data:
253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C5C2E0F5F827DFC

Request:
882004000001002 53AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C 9000

Example 2 (with CRC):

Command Data:
253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C5C2E0F5F827DFC

Command Data CRC: CEE700A0

Request:
88200102000104253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C
9000

Secure Element Specific APDU Error Codes

This table contains the expected error codes and descriptions that a caller may encounter while working with the Secure Element Applet.

Error Code	APDU Command	Description	Error Code to POS
0x6301	Sign Invoice	PIN verification required before executing a command	1500
0x6302	Verify PIN	PIN verification failed – wrong PIN code	2100
0x6303	Verify PIN	Wrong PIN size	2100
0x6304	Sign Invoice	Maximum number of tax categories exceeded	SDC related
0x6305	Sign Invoice	Secure Element amount has reached the defined limit. The Secure Element is locked and no additional invoices can be signed before the audit is completed.	2210
0x6306	End Audit	End Audit is sent but there is no active Audit	SDC related
0x6307	Sign Invoice	Invoice fiscalization is disabled by system	2210
0x6308	Sign Invoice	Invoice DateTime must be within Certificate validity NotBefore and NotAfter	SDC related
0x6310	Verify PIN	The number of allowed PIN entries exceeded	2110
0x63FF	Sign Invoice	A Secure Element counter has reached its limit. The Secure Element must be replaced.	SDC related
0x6700	End Audit	Data must be 256 bytes long	SDC related

0x6A80	End Audit	Proof of Audit command payload provided as APDU Command Data does not match the latest Start Audit one which Secure Element expects. Probably a new Start Audit was initiated after this one was ended.	SDC related
0x6A80	Sign Invoice	The tax category order id exceeds the maximum allowed for the Secure Element.	2310
0x6F00	End Audit	APDU Command Data cannot be recognized as a valid Proof of Audit	SDC related

File-Based Communication

Introduction

This section contains the description of the **File-based** communication with E-SDC.

File-based communication between TaxCore.API and E-SDC is foundation of Local Audit process.

General structure for storing files on removable memory units:

Removable memory root

- UID (folder)
  - audit packages (file)
  - UID.arp (file)
- UID.commands (file)
- UID.results (file)

NOTE:
The above structure displays files for all individual file transfers (copy audit files from E-SDC, upload commands to E-SDC, and transfer commands result from E-SDC). It is used here for explanatory purposes. Each individual file transfer includes only the files specific for that transfer.

Content

1.

SD Cards or Flash Memory Drives Format
Each E-SDC shall work with the following file system formats of SD Cards and USB Flash drives:

2. E SDC is Configured Using Initialization Commands from an SD Card
   JSON file containing all pending commands must be stored in the root of the external disk volume and named **{UID}.commands** (e.g `D:\\YJ37C9Z9.commands`)

3. E SDC Stores Audit Files on SD Card or USB Drive
   An E-SDC shall perform an audit automatically once an SD Card or USB drive is inserted. If any commands are received on the same medium, they shall be executed **before** the proceeding with the Local audit.

4. E SDC Executes Commands Received via SD Card or USB Drive
   An E-SDC shall process commands automatically upon insertion of SD Card or USB Flash drive. Command execution takes precedence over a Local audit.

5. E SDC Stores a Command Execution Result to the SD Card or USB Drive
   After commands have been executed, E-SDC must generate a JSON file named **{UID}.results**. The result must be in format described in paragraph *Commands Execution Results* in Commands. It is stored in the root folder on the SD Card/USB drive.

SD Cards or Flash Memory Drives Format

Each E-SDC shall work with the following file system formats of SD Cards and USB Flash drives:

- FAT
- FAT32
- NTFS

E-SDC is Configured Using Initialization Commands from an SD Card

Commands File

JSON file containing all pending commands must be stored in the root of the external disk volume and named **{UID}.commands** (e.g `D:\\YJ37C9Z9.commands`)

Command file may be generated and downloaded from either Taxpayer Admin Portal (by taxpayers) or the

internal back office portal (by tax authority officers).

Commands File Format

Command file content must be formatted as valid JSON file.

```
[
    {
      "commandId": "GUID",
      "type": 0,
      "payload": "Command Specific Json as string",
      "uid": "string"
    }
]
```

End Of Line Characters

E-SDC must be able to process commands file using any standard EOL characters.

E-SDC Stores Audit Files on SD Card or USB Drive

An E-SDC shall perform an audit automatically once an SD Card or USB drive is inserted. If any commands are received on the same medium, they shall be executed **before** the proceeding with the Local audit.

NOTE:
Audit files must remain stored in E-SDC's local memory until it receives a Proof-of-Audit command from TaxCore.API

All files shall be stored in the folder(s) named after the UID(s) of the secure element(s) which created them.

Example: `G:\BJ3PN1S9\`, where G is the root of the SD card/USB drive and contains audit packages created by the secure element with the UID BJ3PN1S9.

If the folder(s) do not exist, an E-SDC shall create new one(s) - one for each UID.

All audit package files stored in E-SDC's local memory, no matter which secure element was used to create them shall be copied to the appropriate folder(s). Audit package files must be named using the following convention: `{UID}-{UID}-{Ordinal_Number}.json`.

Depending on whether the smart card secure element is connected to the E-SDC, the folder named after its UID

may contain one `{UID}.arp` file. The content of the `{UID}.arp` file is described in Submit Audit Request Payload - ARP.

File structure for this transfer should look like this:

Removable memory root

- UID folder
  - audit package files
  - UID.arp

NOTE:
Dumped audit package files must be created by secure element(s) from the appropriate target environment, e.g. if the Local Audit is performed for the production environment, only audit package files created by the production environment secure elements can be dumped on an SD Card or USB Drive. This is done in order to avoid uploading audit packages to environments which do not recognize the UIDs created for different environments.

E-SDC Executes Commands Received via SD Card or USB Drive

An E-SDC shall process commands automatically upon insertion of SD Card or USB Flash drive. Command execution takes precedence over a Local audit.

JSON file containing all pending commands must be stored in the root of the external disk volume and named **{UID}.commands** (e.g `D:\\YJ37C9Z9.commands`).

File structure for this transfer should look like this:

Removable memory root

- UID.commands

E-SDC shall execute only those commands with the same UID as UID assigned to the digital certificate of the Secure Element (stored in the *SerialNumber* field of the certificate subject).

Command types and the structure are explained in section Commands.

E-SDC Stores a Command Execution Result to the SD Card or USB Drive

After commands have been executed, E-SDC must generate a JSON file named **{UID}.results**. The result must be in format described in paragraph *Commands Execution Results* in Commands. It is stored in the root folder on the SD Card/USB drive.

Example: **G:\BJ3PN1S9.results**

where **G is the root of the SD card/USB drive** and **BJ3PN1S9 is an example UID** of the smart card in use

If file with the same name exists on the SD Card/USB drive, it must be overwritten.

File structure for this transfer should look like this:

Removable memory root

- UID.results