Radio Frequency Identification Chip Reader

Software Development Kit
Version 3.5

Programmer's Guide

RGVI.00010-01 33 01

Regula 2021

Edition from 22.12.2020
Version 3.5.60.120

CONTENTS

LIST OF ABBREVIATIONS

AA – **Active Authentication** – a procedure of additional verification of document authenticity (compliance of the read SO with the original chip)

AT – **Authentication Terminal** – terminal type

BAC/BAP –**Basic Access Control/Protection** – data access control security mechanism

CA – **Chip Authentication** – a stage of an advanced security mechanism of data access control (EAC)

CAN – **Card Access Number** – a short password printed on the document; it is used as a key to control access to protected data

CHAT – **Certificate Holder Authorization Template** – data object containing identifier of terminal type and combination of flags of access rights to informational and functional capabilities of electronic document, delegated to the terminal by the superior subject

DO – Document Owner

EA – **Effective Authorization** – determination of combination of access rights to informational and functional capabilities of the electronic document according to the results of terminal authentication procedure

EAC/EAP –**Extended Access Control/Protection** – advanced security mechanism of data access control

EPROM – Erasable Programmable Read-Only Memory

eSign-PIN – **Personal Identification Number for *eSign* Application** – a short secret password, which is known only to the document holder; it is used as a key to the function of digital signature generation

IS – **Inspection System** – terminal type

LDS – **Logical Data Structure**

MCL – SDK **Main Control Library**

MRZ – document **Machine Readable Zone** used as a key to control access to protected data

OCR – **Optical Character Recognition**

OS – Operating System

PA – **Passive Authentication** – a security mechanism of RFID-chip data integrity verification

PACE – **Password Authenticated Connection Establishment** – data access control security mechanism

PC – Personal Computer

PIN – **Personal Identification Number** – a short secret password known only to the document holder; it is used as a key to control access to protected data

PKD – **Public Key Directory**

PUK – **PIN Unblock Key** – a long secret password known only to the document holder; it is used as a key to control access to protected data

RFID – **Radio Frequency Identification**

RFID-chip – Radio frequency identification chip

RI – **Restricted Identification** – a procedure of chip identification within the context of a certain terminal sector

SAC – **Supplemental Access Control** – an advanced security mechanism of data access control using PACE as a basic mechanism of SM

SAI – **Scanning Area Identifier** – a password printed on the document (as a text field, a bar-code or a special MRZ); it is used as a key to control access to protected data

SDK – **Software Development Kit**

SM – **Security Messaging** – a mechanism of protected data exchanging

SO – **Security Object** – an object of electronic document data protection

ST – **Signature Terminal** – terminal type

SW – Software

TA – **Terminal Authentication** – a stage of an advanced security mechanism of data access control (EAC)

UT – **Unauthenticated Terminal** – terminal type

REFERENCES

[1] BSI. Technical Guideline: Advanced Security Mechanisms for Machine Readable Travel Documents – Extended Access Control (EAC), Version 1.11. TR-03110, 2008.

[2] ICAO. Machine Readable Travel Documents – Part 1, Volume 2: Machine Readable Passports, Specifications for electronically enabled passports with biometric identification capabilities, ICAO Doc 9303, 2006.

[3] ICAO. Machine Readable Travel Documents – Part 3, Volume 2: Machine Readable Official Travel Documents, Specifications for electronically enabled official travel documents with biometric identification capabilities, ICAO Doc 9303, 2008.

[4] ITU-T. Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). X.690, April 2002.

[5] RFC 3279. W. Polk, R. Housley, L. Bassham, «Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile», April 2002.

[6] RFC 5280. R. Housley, W. Polk, W. Ford, D. Solo, «Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile», May 2008.

[7] RFC 3852. Cryptographic Message Syntax (CMS), July 2004.

[8] RFC 3447. J. Jonsson, B. Kaliski, «Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications, Version 2.1», February 2003.

[9] RSA Laboratories. PKCS #3: Diffie-Hellman Key Agreement Standard. RSA Laboratories Technical Note, Version 1.4, 1993.

[10] RSA Laboratories. PKCS #8: Private Key Information Syntax Standard. RSA Laboratories Technical Note, Version 1.2, 1993.

[11] BSI. Technical Guideline TR-03111: Elliptic Curve Cryptography (ECC) based on ISO 15946, Version 1.0, 2007.

[12] BSI. Technical Guideline TR-03105, Part 5.1: *ePassport* Conformity Testing. Test plan for ICAO compliant inspection systems with EAC. Version 1.2, 11.09.2009.

[13] ISO/IEC 19794-2:2005, Information technology – Biometric Data interchange formats – Part 2: Finger minutiae data.

[14] ISO/IEC 19794-4:2005, Information technology – Biometric Data interchange formats – Part 4: Finger image data.

[15] ISO/IEC 19794-5:2005, Information technology – Biometric Data interchange formats – Part 5: Face image data.

[16] ISO/IEC 19794-6:2005, Information technology – Biometric Data interchange formats – Part 6: Iris image data.

[17] NISTR 6529. Common Biometric Exchange File Format (CBEFF).

[18] ISO/IEC 14443-3, Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 3: Initialization and anti-collision.

[19] ISO/IEC 14443-4, Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 4: Transmission protocol.

[20] ISO/IEC 7816-4, Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange.

[21] PC/SC Workgroup. Interoperability Specification for ICCs and Personal Computer Systems. Revision 2.01, September 2005.

[22] NXP Semiconductors. Application Note: MIFARE Type Identification Procedure. Revision 3.6, July 2016.

[23] ICAO Technical Report: Supplemental Access Control for Machine Readable Travel Documents. Version 1.1, April 15, 2014.

[24] BSI. Technical Guideline TR-03110: Advanced Security Mechanisms for Machine Readable Travel Documents – Part 1: "eMRTDs with BAC/PACEv2 and EACv1", Part 2: "Extended Access Control (EACv2), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI)", Part 3: "Common Specifications", Version 2.10, 2012.

[25] BSI. Technical Guideline TR-03127: Architecture electronic Identity Card and electronic Resident Permit. Version 1.13, 2011.

[26] BSI. Technical Guideline TR-03117: eCards mit kontaktloser Schnittstelle als sichere Signaturerstellungseinheit. Version 1.0, 2009.

[27] BSI. Technical Guideline TR-03105, Part 5.2: *ePassport* Conformity Testing. Test plan for *eID* and *eSign* compliant eCard reader systems with EAC 2. Version 1.1, 11.05.2011.

[28] ISO/IEC 9796-2:2002, Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms.

[29] ISO/IEC 10118-3:2003, Information technology – Security techniques – Hash functions – Part 3: Dedicated hash functions.

[30] ISO/IEC 7816-4:2005, Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange.

[31] ICAO. Technical Report – LDS and PKI Maintenance. Version 2.0, May 21, 2014.

[32] RFC 2849. G. Good. The LDAP Data Interchange Format (LDIF) - Technical Specification. June, 2000.

[33] http://www.icao.int/Security/mrtd/Pages/icaoPKD.aspx

[34] ICAO. Technical Report – CSCA countersigning and Master List issuance. Version 1.0, June 23, 2009.

[35] ICAO. Supplement to Doc 9303. Release 14, May 13, 2014.

[36] BSI. Technical Guideline TR-03129: PKIs for Machine Readable Travel Documents. Version 1.10, 2009.

[37] ISO/IEC 18013-2, Information technology – Personal identification – ISO-compliant driving licence – Part 2: Machine-readable technologies.

[38] ISO/IEC 18013-3, Information technology – Personal identification – ISO-compliant driving licence – Part 3: Access control, authentication and integrity validation.

[39] Commission regulation (EU) No 383/2012 of 4 May 2012 laying down technical requirements with regard to driving licences which include a storage medium (microchip).

[40] BSI. Technical Guideline TR-03129-2: PKIs for Machine Readable Travel Documents. Protocols for the Management of Certificates and CRLs – National Protocols for ePassport Application Version 1.12, 2016.

INTRODUCTION

The present «*Programmers Guide*» describes the order of SDK tools usage when developing user applications for work with «Regula» devices equipped with RFID-chip reader.

SYSTEM REQUIREMENTS

Minimum:

- CPU .... Pentium IV 2.0 GHz
- RAM .... 512 MB
- OS .... Windows 2000 (Service Pack 4), Windows XP (Service Pack 1)
- SystemBus .... built-in USB 2.0 Hub with full High Speed mode support

Recommended:

- CPU .... Pentium IV (Duo Core) 3.0 GHz or higher
- RAM .... 1 GB and more
- OS .... Windows 7
- SystemBus .... built-in USB 2.0 Hub with full High Speed mode support

WHAT'S NEW?

Version 3.5:
- transition to the use of OpenSSL cryptographic libraries;
- full support for *eDL* application data access.

Version 3.4:
- changes made in exported function types (parameters) and SDK structure (all COM-components removed), in notification callback-function mechanism (being executed in the inner SDK threads context, main application is responsible for the necessary synchronization).

Version 3.3:
- provided support for the access to *eDL* application data without EAP mechanism use (CA, TA) .

Version 3.2:
- full support of Integrated Mapping and Chip Authentication Mapping modes of PACE.

Version 3.1:
- added scenario-based mechanism for session work;
- extended format for XML-representation of results.

Version 3.0:
- added functionality to provide full support for EAC (version 2) adn SAC advanced access mechanisms;
- provided full support for *eID*, *eSign* application data access and their functionality;
- implemented support for session work with RFID-chips, providing maximum flexibility for the construction of software logics.

Version 2.1:
- support of national character sets – transfer to representation of text data in `UTF8` format.

Version 2.0:
- transition to work according to PC/SC protocol, device driver update;
- EAC support;
- extended analysis of the read data structure and their compliance with standards;
- support for data reading using the extended length commands.

Version 1.4:
- certificate verification added for the passive authentication (general PKD support);
- changes in **`TRF_Authentification`** and **`TRF_SOD_Certificate`** data structures.

Version 1.2:

- added the possibility of the passive/active authentication.

Version 1.0:

- first SDK version.

1. SDK STRUCTURE

\<Program Files>\Regula\RFID Reader SDK:

`RFID_SDK.dll`	–	SDK MCL;
`RFID_SDK_UI.dll`	–	library for the composition of working XML-scenario;
`Imaging.dll`	–	image graphics formats support library (JPG, JPEG-2000, TIFF, PNG, WSQ, BMP);
`RFIDtest2.exe`	–	test application project executable module;
`RFIDtest3.exe`	–	demo program;

\<Program Files>\Regula\RFID Reader SDK\FirmwareUpdate – utility for device EPROM re-programming;

\<Program Files>\Regula\Samples\RFID SDK\C++ – directory of `RFIDtest2` test application project, illustrating the use of SDK software tools for working with electronic document in batch mode (`C++ Builder`);

\<Program Files>\Regula\RFID Reader SDK\Doc:

`Programmers Guide (en).pdf`	–	this Guide in English;
`Programmers Guide (ru).pdf`	–	this Guide in Russian.

2. SDK FEATURES

SDK allows:

- reading data from RFID-chip memory (international standard ISO/IEC 14443) when working with «Regula» devices equipped with RFID-chip reader;
- performing procedures of passive and active authentication of travel document based on data read in accordance with the requirements of [2], [3].

SDK provides:

- access to protected data of RFID-chip with an automatic application of BAC/BAP and EAC (version 1.11) procedures when the user presents all the required additional information (MRZ lines, a set of certificates corresponding to the read document) [1];
- control of read data integrity, compliance of it format and contents with the requirements of the respective normative documents.

Working with electronic document in a session mode the following is ensured:

- possibility of organizing logic of the software for the implementation of various authentication procedures (*Standard Inspection Procedure*, *Advanced Inspection Procedure*, *General Authentication Procedure*) with full support for the use of BAC, PACE, EAC (versions 1.11 and 2) [23, 24];
- access to `eID` application data and a possibility to use its additional built-in functions [25];
- possibility to use functions of `eSign` application [26];
- access to `eDL` application data [37, 38, 39].

> *Note.* Starting with SDK version 2.0, «Regula» 7051 RFID-chip reader operates under control of PC/SC-driver. Data exchange between the reader and a RFID-chip is performed according to the specification [21]. Thus, RFID-reader control is unified, so it is possible to use not only the means of the SKD to work with it, but third-party software as well.

3. INSTALLATION AND USE OF SDK TOOLS

SDK for RFID-chip readers is included in the *«SDK for Document Readers "Regula" Mod. 70x3.xxx, 70x4.xxx, 83x3»*, starting with version 4.3 of the latter.

When installing the SDK an installation of all required program components is performed, including RFID-chip reader driver.

When connecting the device into available USB slot the operational system (OS) will notify you about the detection of the new device and will activate the driver.

To use the SDK software in the user's project it is required to:

- include **`RFID.h`** and **`PasspR.h`** header files with descriptions of functions exported from **`RFID_SDK.dll`** and **`RFID_SDK_UI.dll`**, the used data structures and constants, or to replace them with respective declarations (if the application is developed not on `C++`);
- dynamically connect **`RFID_SDK.dll`** and **`RFID_SDK_UI.dll`** control libraries, get the pointers to the exported functions using Windows API **`GetProcAddress()`** function.

The path to the control libraries is registered when installing SDK in the system registry in **`Path`** string value of «**`HKEY_CURRENT_USER\SOFTWARE\Regula\RFID Reader SDK`**» key. The SDK version is specified in *Version* string value of the same key.

4. GENERAL INFORMATION

4.1. RFID-CHIP TYPES

The RFID-chips are divided into several types according to the following criteria:

1. **Physical parameters of the connection between chip and reader antennas** (ISO/IEC 14443-2):
   - type `A`;
   - type `B`.

2. **Communications protocol**:
   - ISO/IEC 14443-3 (MIFARE® Classic Protocol) (for type A);
   - ISO/IEC 14443-4 (for types A and B).

3. **Data protection method** (for chips with ISO/IEC 14443-4 support):
   - with data protection using SM [2, 3, 24], EAC/EAP [1, 24];
   - without data protection [20].

The «Regula» 7051 reader provides data reading from the memory of RFID-chips of all the above-listed types.

4.2. Logical Data Structure of RFID-chips (Protocol MIFARE® Classic Protocol)

Memory of RFID-chips supporting the communication protocol by the standard ISO/IEC 14443-3 (MIFARE® Classic Protocol), like MIFARE® 1K, MIFARE® 4K, MIFARE® Ultralight, has a definite logical structure. It is divided into sectors, each of which is in turn divided into blocks. The size of one block of data is fixed and is 16 bytes. The sector size varies depending on the total chip memory amount and sector's location in it. The first 32 memory sectors of any chip consist of 4 blocks (64 bytes). All the subsequent sectors consist of 16 blocks (256 bytes).

Thus, for instance, for the chip MIFARE® 1K with memory volume of 1 Kb, all memory will be divided into 16 sectors with 4 blocks in each of them ($16 \cdot 4 \cdot 6 = 1024$ bytes). For the chip MIFARE® 4K with memory size of 4 Kb – the first 32 sectors will consist of 4 blocks each, and the sectors from the 33rd to the 40th – of 16 ($32 \cdot 4 \cdot 16 + 8 \cdot 16 \cdot 16 = 4098$ bytes).

4.3. Logical Data Structure of RFID-chips (protocol ISO/IEC 14443-4)

From a software standpoint, the data contained in the memory of RFID-chip are organized in a form of separate files. Each file has its own unique identifier, that is used to provide access to the file. The logical designation of files (of their contained data) is defined by the application, which includes the file. Each application also has a unique identifier that is used to select the application. A separate application can provide a range of informational and (or) functional capabilities.

The file that is not included in any of the applications is considered to belong to the root *Master File*.

4.3.1. ePassport Application

When implementing the requirements of ISO/IEC 14443-4 for travel documents with embedded RFID-chip, the logical data structure of `ePassport` application is defined by the documents [2], [3], [23], [31].

There are *service* and *informational data groups* (*files*).

The **service data groups** include:

- `EF.COM` – information about presence of informational data groups;
- `EF.SOD` – data of the electronic document security object: digital signature and other information used for passive authentication procedure of travel document;
- `EF.CVCA` – public key identifier required for performance of TA procedure for EAC (version 1.11).

When implementing the requirements of [23], [24] *mandatory* `EF.CardAccess` is added to the chip service data groups. It contains information about the algorithms and the order of performing the procedures of secure data access (PACE as a basic SM procedure, EAC version, CA and TA algorithms).

`EF.CardAccess` file is not included in the structure of `ePassport` and it is located in the root *Master File*.

The **informational data groups** include:

- `EF.DG1` – *mandatory* group containing MRZ data;
- `EF.DG2` – facial *biometric* graphic data of the DO [15];
- `EF.DG3` – fingerprint *biometric* graphic data of the DO [14];

- EF.DG4 – iris *biometric* graphic data of the DO [16];
- EF.DG5 – additional photos of the DO;
- EF.DG6 – reserved for further standard development;
- EF.DG7 – image of DO's signature;
- EF.DG8 – reserved for further standard development;
- EF.DG9 – reserved for further standard development;
- EF.DG10 – reserved for further standard development;
- EF.DG11 – additional personal details of the DO;
- EF.DG12 – additional information about the document;
- EF.DG13 – additional details (reserved for use by the national services of the issuing state);
- EF.DG14 – information about cryptographic algorithms and keys used when implementing EAC mechanism (performing CA and TA procedures);
- EF.DG15 – information about active authentication public key;
- EF.DG16 – information about persons to notify in case of emergency.

DG2, DG3 and DG4 groups contain biometric information in format specified by the ISO/IEC 7816-11, which is compatible with the requirements of «*The Common Biometric Exchange Formats Framework*» (*CBEFF*) [17].

This format allows storing a single data structure of multiple *records* of the same type of biometric data (photos, fingerprints' images etc.). Each record type is defined by its header. The format of each specific type of record may provide the storing of several *templates* of a certain type of biometric data (for example, fingerprints of different fingers or different images of the iris). In turn, each of the templates can be represented by several *variants* of images (for example, several versions of a fingerprint of the same finger).

In the case of data protection with the use of SM (BAC or PACE) file access (except EF.CardAccess) is done using special procedures of secure data exchange between the reader and the RFID-chip in compliance with the order of their conduct and the requirements of the specifications [2], [3] and [24].

When biometric data from DG3 and DG4 are additionally protected using EAC mechanisms, the access to these groups is done according to the specifications [1] (EAC version 1.11) or [24] (EAC version 2).

4.3.2. eID Application

When implementing the requirements of ISO/IEC 14443-4 for identification card with embedded RFID-chip the logical data structure of *eID* application is defined by the document [25].

The **service files** include:

- `EF.CardAccess` – *mandatory* file containing information about the algorithms and the order of performing of secure data access procedures (PACE as a basic SM procedure, EAC version, CA and TA algorithms);
- `EF.CardSecurity` – *mandatory* file containing information about the algorithms and the order of performing of secure data access procedures (CA keys parameters) and additionally built-in functions (RI), digital signature and other information used for the document passive authentication procedure (file is a SO);
- `EF.ChipSecurity` – *optional* file containing information about the algorithms and the order of performing of secure data access procedures (chip-specific CA keys parameters) and additional built-in functions (RI), digital signature and other information used for the document passive authentication procedure (file is a SO).

These files are not included in the structure of *`eID`* and are located in the root *Master File*.

The **informational data groups** of *`eID`* application include:

- `EF.DG1` – document type;
- `EF.DG2` – code of issuing state;
- `EF.DG3` – document date of expiration;
- `EF.DG4` – DO's name;
- `EF.DG5` – DO's surname;
- `EF.DG6` – religious/artistic name (alias) of the DO;
- `EF.DG7` – DO's academic title;
- `EF.DG8` – DO's date of birth;
- `EF.DG9` – DO's place of birth;
- `EF.DG10` – DO's nationality;
- `EF.DG11` – DO's sex;
- `EF.DG12` – additional details;
- `EF.DG17` – DO's place of residence;
- `EF.DG18` – DO's personal identifier (`Community ID`);
- `EF.DG19` – details about permanent residence permit (1);
- `EF.DG20` – details about permanent residence permit (2);
- `EF.DG21` – additional details.

`EF.DG13`–`DG16` data groups are reserved for further standard development [25].

In the case of data protection with the use of SM (PACE) file access (except `EF.CardAccess`) is done using special procedures of secure data exchange between the reader and the RFID-chip in compliance with the order of their conduct and the requirements of the specifications [24].

As the additional built-in functions `eID` application provides features:

- **restricted identification** (**RI**) of the chip;
- **verification** of *DO's* `Community ID` and *DO's age* by means of a chip.

4.3.3. eSign Application

Logical data structure for `eSign` application is defined by the document [26].

The **service files** include:

- `EF.CardAccess` – *mandatory* file containing details about the algorithms and the order of performing of secure data access procedures (PACE as a basic SM procedure, EAC version, CA and TA algorithms).

`EF.CardAccess` is not included in the structure of `eSign` and is located in the root *Master File*.

`eSign` application grants the user with the access to the function of data digital signature generation only after the General Authentication Procedure (see section 4.11.3).

To open an access to `eSign` application functions after the document personalization procedure the application activation is performed including creation of:

- pair of cryptographic keys;
- password of access to the function of the digital signature generation (*eSign-PIN*).

To close an access to the application it is required to terminate the active key pair and (or) active *eSign-PIN*.

Usually `eSign` is used in conjunction with other applications, such as `eID`, providing additional functionality to the identification document.

4.3.4. eDL application

When implementing the requirements of ISO/IEC 14443-4 for driving licenses with embedded RFID-chip the logical data structure of `eDL` application is defined by the documents [37], [38], [39].

There are *service* and *informational data groups* (*files*).

The **service data groups** include:

- `EF.COM` – information about presence of informational data groups;

- `EF.SOD` – data of the electronic document security object: digital signature and other information used for passive authentication procedure of driving license;

The **informational data groups** include:

- `EF.DG1` – *mandatory* group containing demographic data elements and vehicle categories/restrictions/conditions;
- `EF.DG2` – optional license holder information;
- `EF.DG3` – optional issuing authority details;
- `EF.DG4` – optional portrait image(s);
- `EF.DG5` – optional signature/mark image(s);
- `EF.DG6` – optional facial *biometric* template(s);
- `EF.DG7` – optional finger *biometric* template(s);
- `EF.DG8` – optional iris *biometric* template(s);
- `EF.DG9` – optional other biometric;
- `EF.DG10` – reserved for future use;
- `EF.DG11` – optional domestic data (reserved for domestic use, the encoding is defined domestically);
- `EF.DG12` – non-match alert – reaction on detection of any differences between the machine-readable information and the human-readable information (printed on a document);
- `EF.DG13` – information about active authentication public key;
- `EF.DG14` – information about cryptographic algorithms and keys used when implementing EAP mechanism (performing CA and TA procedures).

`DG6`, `DG7` and `DG8` data groups are analogues of `DG2`, `DG3` and `DG4` data groups of *`ePassport`* application (see section 4.3.1).

In the case of data protection with the use of SM (BAP) file access is done using special procedures of secure data exchange between the reader and the RFID-chip in compliance with the order of their conduct and the requirements of the specifications [37], [38], [39].

When biometric data from `DG7` and `DG8` are additionally protected using EAP mechanisms, the access to these groups is done according to the specifications [38].

4.4. Access Keys to Protected Data

To establish a secure communication channel between the reader and the RFID-chip when implementing SM mechanism several types of keys (passwords) can be used. Each of them can be used by the terminal of a certain type.

The set of functional and informational capabilities ensured by the chip also depends on the type of the used password:

- **MRZ** – data access password is derived from MRZ printed on the document and available for OCR operation.

For IS terminal this type of password provides its full functionality; for other types it is not used.

MRZ can be used to organize the SM communication channel using both BAC and PACE as a basic mechanism.

- **SAI** – a password, usually printed on the document as a separate text field, a bar-code or a special MRZ and available for OCR operation.

For IS terminal this type of password provides its full functionality; for other types it is not used.

SAI can be used to organize the SM communication channel using BAP (for *eDL* application).

- **CAN** – a short password, usually printed on the document and available for OCR operation.

For IS and AT terminals this type of password provides their full functionality; for ST – possibility to create a digital data signature, as well as change of *eSign-PIN*; for UT it is not used.

CAN can be used to organize the SM communication channel using PACE as a basic mechanism only.

- **PIN** – a short secret password known only to the DO.

For AT terminal this type of password provides its full functionality; for ST – possibility of creation/termination of *eSign-PIN* and generation/termination of cryptographic key pair; for IS it is not used.

PIN can be used to organize the SM communication channel using PACE as a basic mechanism only.

- **PUK** – a long secret password known only to the DO.

For AT, ST and UT terminals this type of password provides a possibility of unblocking the PIN and the *eSign-PIN*; for IS it is not used.

PUK can be used to organize the SM communication channel using PACE as a basic mechanism only.

4.5. Password Management

4.5.1. PIN

During work with electronic document performing PACE procedure situations might arise when:

- it is necessary to change PIN (or CAN) value;
- PIN is temporary suspended (after failed attempts of its application the counter of the remaining attempts has reached `1`);
- PIN is blocked (after failed attempts of its application the counter of the remaining attempts has reached `0`);
- it is required to activate PIN, because the *eID* application is deactivated;
- it is required to deactivate the PIN for deactivation of *eID* application.

All these situations require a certain set of possible actions from the user to be able to run correctly with the document. They include the following operations:

- change CAN;
- change PIN;
- resume PIN ;
- unblock PIN;
- activate PIN ;
- deactivate PIN.

To **change PIN/CAN** it is required to:

- perform General Authentication Procedure (see clause 4.11.3) using the current PIN/CAN;
- perform a separate operation of password change.

To **resume PIN** it is required:

- to perform PACE procedure using the CAN for temporary resuming the current PIN;
- within the context of the secure communication channel established in the previous step to perform PACE procedure using temporary resumed PIN.

To **unblock PIN** it is required to:

- perform PACE procedure using the PUK;
- perform a separate operation of password unblocking.

To **activate/deactivate PIN** it is required to:

- perform General Authentication Procedure;
- perform a separate operation of password activation/deactivation.

All the above-listed operations (except PIN unblocking) are available only for a terminal with effective AT type and *PIN Management* authorized right (see section 4.7).

4.5.2. eSign-PIN

The operations of management of *eSign-PIN* password (used for access to the function of data digital signature generation of the `eSign` application) can be attributed to a separate category.

During work with electronic document situations may arise when:
- it is required to create *eSign-PIN*, because `eSign` application has not been yet initialized or it was deactivated;
- it is required to change the value of *eSign-PIN*;
- *eSign-PIN* is blocked (after failed attempts of its application the counter of the remaining attempts has reached `0`);
- it is required to terminate *eSign-PIN* for deactivation of `eSign` application.

To **create/destroy *eSign-PIN*** it is required to:
- perform General Authentication Procedure using PIN;
- perform a separate operation of creation/destruction of *eSign-PIN*.

To **unblock *eSign-PIN*** it is required to:
- perform General Authentication Procedure using PUK;
- perform a separate operation of unblocking of *eSign-PIN*.

To **change *eSign-PIN*** it is required to:
- perform General Authentication Procedure using CAN;
- perform a separate operation of changing *eSign-PIN* supposing additional indication of the current password value.

All the above-listed operations are available only for a terminal with effective ST type.

4.6. Terminal Types

The **terminal** means a personal computer (PC) on which the software uses the SDK tools to communicate with the RFID-chip.

There are several types of terminals. The SDK provides support of a separate set of functional capabilities for each of them:

- **`Inspection System (IS)`**.

This type of terminal provides access to the data groups of *`ePassport`* and *`eDL`* applications with a possibility to perform verification of document authenticity.

This type of terminal also provides access (only for reading) to the data groups of the *`eID`* application.

- **`Authentication Terminal (AT)`**.

This type of terminal provides access to the data groups of the *`eID`* application (reading or updating) with a possibility to perform additional procedures of RI and auxiliary data verification, as well as a possibility of data access key management (change PIN/CAN, PIN blocking).

For the *`eSign`* application a possibility is provided for installation of *eSign-PIN* and generation of a new pair of cryptographic keys for use in operations with digital data signature.

- **`Signature Terminal (ST)`**.

This type of terminal provides access to the function of the data digital signature generation of *`eSign`* application.

- **`Unauthenticated Terminal (UT)`**.

This type of terminal provides access to the functions of the new PIN installation (for *`eID`*) and unblocking of PIN and *eSign-PIN*.

In terms of the RFID-chip, the terminal is «*unauthenticated*» until the successful completion of terminal authentication.

4.7. EFFECTIVE TERMINAL AUTHORIZATION

The process of determination of a combination of protected data access rights and functional capabilities of electronic document for the current terminal is called **effective terminal authorization**. The main technical element of this process is the procedure of **terminal authentication** (*Terminal Authentication*, *TA*).

On the one hand, TA is reduced to verification of the digital signature of the control data fragment generated in accordance with specific requirements [1], [24].

The public key, with the help of which the chip must perform verification of the transmitted digital signature, is contained in a special *terminal certificate*.

The terminal certificates are released by the subject called the *Document Verifier* (*DV*) – organizational sub-division administering one or another group of terminals. The contents of the terminal certificate are signed by a digital signature, which allows verifying the authenticity of this certificate. Verification of digital signature of the terminal certificate is possible only if there is another certificate containing a respective public key– of *DV-certificate*.

DV-certificates are released by the basic subject – *Country Verifying Certificate Authority* (*CVCA*) – and are signed also by a digital signature for provision of their authentication. Digital signature of DV certificates are verified by using the public key, which is recorded in the chip private memory at stage of electronic document production.

It is possible also to switch to using of a different CVCA-key for verification of DV-certificate signature with the help of a special *CVCA-Link-certificate*.

Data of all three above-listed types of certificates (CVCA-Link-, DV- and terminal certificate) are represented in a definite format to verify them by RFID-chip (*Card Verifiable*, *CV*). They contain:

- own digital signature;
- public key identifier, which is necessary for verification of certificate digital signature (*Certificate Authority Reference*, *CAR*);
- own identifier (i. e. identifier of public key containing in the certificate) (*Certificate Holder Reference*, *CHR*);
- public key data.

Thus, knowing the CVCA public key identifier, it is possible to generate a respective certificate chain and to verify the authenticity of each of them consequently, completing straight with verification of digital signature of the data control fragment. This is the technical aspect of TA.

On the other hand, each of the above certificate contains not only the data associated with the implementation of TA's technical side, but also another important component – a *combination of access rights* to protected data and functionality of electronic document delegated by the superior entity to the subordinate one (for example, from DV to one or another terminal). This information is stored in a special CHAT certificate data object (*Certificate Holder Authorization Template*).

It is the formation of a logical combination of delegated rights from the entire given certificate chain that makes the process of effective terminal authorization. In fact, the terminal confirms its rights by having access to the respective certificates, as well as to the private cryptographic key, that used for a generation of data control fragment digital signature that need to be send to the chip to be verified.

During EAC (version 2) the delegated by the superior subjects combination of access rights may be additionally restricted on the level of the terminal, i.e., straight by the user. This is achieved by declaring the information about the terminal type and the required combination of access rights at the stage of terminal configuration when preparing the procedure of SM-communication channel opening (see clauses 5.8.4, 5.8.7), which, as the CHAT data object, is transferred to the chip during PACE procedure [24, part 3, §B.11.1].

The type of protected data access that is used during PACE procedure influences the final result of effective authorization as well.

For various terminal types the possible set of delegated access rights differs (Table 1). In compliance with it the format of the data object differs as well, which serves for representing such combination of rights [24, part 3, §C.4].

The set of `ePassport` application read access rights for **IS terminal**:

- `DG3`;
- `DG4`.

In addition, read access is guaranteed by default for IS terminal for all information groups of data of the `eID` application.

The set of access rights for **AT terminal**:

- rights to read the data of `eID` application informational groups (`DG1`–`DG21`) that are defined separately;
- rights to write (update) the data of `eID` application informational groups (`DG17`–`DG21`) that are defined separately;
- *Install Qualified Certificate* – right to generate a pair of cryptographic keys for `eSign` application;
- *PIN Management* – right to use PIN password management instructions;
- *CAN Allowed* – right to use CAN password;
- *privileged Terminal* – right to use *privileged* CA keys;
- *restricted Identification* – right to perform restricted identification;
- *community ID Verification* – right to perform verification of `Community ID`;
- *age Verification* – right to perform the user age verification.

Access right for **ST terminal**:

- performing the operation of data digital signature generation.

Table 1

Summary table for the access rights to the resources of electronic document for different terminal types

Operation		IS	AT	ST	UT
`ePassport, eDL` applications					
Data group reading	Not sensitive	**+**	–	–	–
	Sensitive	**EA**	–	–	–
`eID` application					
Data group reading		**+**	**+**	–	–
Data group modification `DG17–DG21`		–	**EA**	–	–
Restricted Identification		–	**EA**	–	–
`CommunityID` verification		–	**EA**	–	–
Age verification		–	**EA**	–	–
`eSign` application					
Key pair generation		–	**EA (PIN)**	–	–
eSign-PIN generation		–	–	**+ (PIN)**	–
eSign-PIN changing		–	–	**+ (CAN)**	–
eSign-PIN unblocking		–	–	**+ (PUK)**	**+ (PUK)**
eSign-PIN terminating		–	–	**+ (PIN)**	–
Key pair terminating		–	–	**+ (PIN)**	–
Digital signature generation		–	–	**EA (CAN)**	–
Password management					
Changing	PIN	–	**EA (PIN)**	–	**+ (PIN)**
	CAN	–	**EA (PIN)**	–	–
Unblocking	PIN	–	**EA (PIN, PUK)**	–	**+ (PUK)**
Activation		–	**EA (PIN)**	–	–
Deactivation		–	**EA (PIN)**	–	–

Note. In brackets the required type of password. EA – is formed by the results of effective terminal authorization.

4.8. Data Security Mechanisms

To protect the data of electronic documents several basic security mechanisms are provided, which are defined in [1], [2], [3], [23], [24]:
- passive authentication (*PA*);
- active authentication (*AA*);
- access control.

4.8.1. Passive Authentication

Passive authentication uses the mechanism of digital signature to confirm the authenticity of data that are stored in RFID-chip memory. It allows detecting the presence of any changes in signed data read from the RFID-chip memory but does not protect against their full copying (cloning of RFID-chip).

Digital signature is generated at the stage of document personalization on the basis of contents of a document security object (SO_D) by the manufacturer of electronic document (so called «*Document Signer*», *DS*). The SO_D itself may contain the hashes (checksums) of data information groups (files) of an application.

To use the digital signature mechanism requires a pair of cryptographic keys. The private key is used to compute the digital signature and is available only for the signer; the public key – to verify the signature value and is distributed as a certificate (a special data object, which is protected by the digital signature mechanism as well).

Thus, the procedure of passive authentication consists of two basic stages to control:
- the authenticity of document security object;
- integrity of document data informational groups.

To verify the authenticity of electronic document with the help of the PA it is required to:
- read SO_D data from the memory of RFID-chip;
- receive DS-certificate with a public key to verify a digital signature of SO_D;
- receive CSCA-certificate (*Country Signing Certificate Authority*) with a public key to verify a digital signature of DS-certificate;
- verify the authenticity of the CSCA-certificate by verification of its digital signature (since it is self-signed, the signature verification may be performed using the public key contained in the certificate itself);
- verify the authenticity of the DS-certificate by verification of its digital signature;
- verify the authenticity of the SO_D by verification of its digital signature;
- verify the authenticity of the read informational data groups by comparing the computed hash values and the corresponding values contained in the SO_D.

Since *Master Lists (ML*) can be used as a storage for CSCA-certificates for SO_D verification [34], a validation of digital signature of master list's security object (SO_{ML}) is a part of passive authentication. This digital signature is generated at the stage of master list issuance on the basis of its contents by the issuer (so called «*Master List Signer*», *MLS*).

To verify the authenticity of master list it is required to:

- receive MLS-certificate with a public key to verify a digital signature of SO_{ML};
- receive CSCA-certificate with a public key to verify a digital signature of MLS-certificate;
- verify the authenticity of the CSCA-certificate by verification of its digital signature (since it is self-signed, the signature verification may be performed using the public key contained in the certificate itself);
- verify the authenticity of the MLS-certificate by verification of its digital signature;
- verify the authenticity of the $SO_{ML}$by verification of its digital signature.

Search for a public key to verify a digital signature can be performed by one of two available criteria:

- a combination of identifier of the source (organization), which has issued the respective certificate (`Issuer`), and the certificate serial number (`serialNumber`);
- identifier of the signature subject (the organization that performed document personalization) (`subjectKeyIdentifier`).

Access to the CSCA-, DS- and MLS- certificates must be provided within the context of the policy of providing the terminal functioning. As a rule, local or centralized certificate storage – *Public Key Directory* (PKD) – is used for these purposes. In most cases the DS-certificate is included directly in the data structure of SO_D; MLS- and corresponding CSCA-certificate can be present in SO_{ML} data structure.

4.8.2. Active Authentication

Active authentication uses mechanism of «challenge – response» to determine the authenticity of RFID-chip.

A pair of cryptographic keys is required for its operation:

- the private key – is stored in protected memory of the RFID-chip and is inaccessible for reading;
- the public key – is stored in a special informational data group `DG15` of *`ePassport`* application (for another applications AA is not provided).

In a process of active authentication, the terminal sends randomly selected data fragment («challenge») to the RFID-chip. The chip generates a digital signature of the data using the private key and returns its value («response») to the terminal. The terminal verifies the validity of the digital signature using the public key, determining thereby the authenticity of the private key used by the chip, and hence the one of the chip itself.

Active authentication allows identifying effectively the fact of RFID-chip cloning.

4.8.3. Access Control

RFID-chip protects the data from unauthorized access by the respective access control mechanisms.

The basis of any access control mechanism is the establishing of a secure communication channel between the reader and the chip (*Security Messaging*, *SM*). At the same time the data to be sent are subject to preliminary encryption and subsequent decryption when received.

In addition to data protection the access control mechanism allows restricting the use of one or another informational or functional chip capabilities by the terminal depending on the specified effective terminal type and delegated access rights.

The data, which are relatively easy to obtain from sources other than the document itself (for example, MRZ, DO photo etc.), are protected by *Basic Access Control/Protection* (*BAC/BAP*).

BAC/BAP only checks that the terminal has physical access to the document by requiring the printed data (MRZ, bar-codes, text fields) to be read optically.

More sensitive personal data (fingerprints, iris) are additionally protected by the extended access control mechanism (*Extended Access Control/Protection*, *EAC/EAP*). Their use is permitted only to authorized terminals, which confirmed their right by successful TA procedure.

4.9. Advanced Security Mechanisms

There are several variants of advanced security mechanisms for electronic document data protection, that are an alternative or supplement of the basic mechanisms:

- *Password Authenticated Connection Establishment* (*PACE*);
- *Chip Authentication* (*CA*);
- *Terminal Authentication (TA*).

If PACE and CA (version 1) may be used as independent protocols for replacement of BAC and AA respectively, then TA may be used only in combination with CA.

4.9.1. Password Authenticated Connection Establishment

When organizing a secure communication channel in [23] and [24] it is proposed to use PACE as BAC/BAP alternative.

Unlike BAC/BAP, stability and security of its cryptographic algorithm directly depends on the key, which is derived from a combination of several fields of MRZ, PACE operates with more durable keys, the «strength» of which does not depend on the «strength» of the used password (CAN or MRZ), which makes this protocol more secure.

4.9.2. Chip Authentication

Chip authentication procedure is one of the components of EAC/EAP. Like BAC/BAP and PACE it serves to organize a secure communication channel, which is more reliable compared to the basic procedures. In addition, CA is an alternative of AA, as it confirms the chip authenticity as well.

CA is based on the use of a static pair of cryptographic keys, which are stored in chip memory.

Implementing EAC/EAP (version 1.11) information about the keys and algorithms of CA is stored in `DG14` data group of *`ePassport`* or *`eDL`* applications.

Implementing EAC (version 2) information about the keys and algorithms of CA is stored in `EF.CardAccess`, `EF.CardSecurity`, `EF.ChipSecurity` files of the root *Master File*.

Successful CA procedure ensures that the public key and the private key stored in the protected chip memory comply with each other. And this in turn confirms that the chip has not been cloned.

4.9.3. Terminal Authentication

The second component of EAC/EAP is the procedure of terminal authentication. TA is also a technical aspect of the process of *effective terminal authorization* (see section 4.7).

To perform TA the following actions are required:

- acquisition of CVCA-key identifier, that is required for DV-certificate authentication. This identifier is contained in a separate service `EF.CVCA` file of *`ePassport`* application (for EAC version 1.11) or is reported by the chip if PACE procedure is successfully performed (for EAC version 2);
- acquisition of the respective CVCA-Link-certificate (if available) and sending to the chip to verify its digital signature and switch to the use of the contained CVCA public key;
- acquisition of the respective DV-certificate and sending to the chip to verify its digital signature with the current CVCA public key;
- acquisition of the respective terminal certificate and sending to the chip to verify its digital signature with the public key from DV-certificate that was previously transmitted;
- generating a digital signature of data fragment, which was formed in compliance with the [1] or [24] (*challenge*), and its transmitting to the chip to verify with the public key from the terminal certificate that was previously transmitted.

Formation of a digital signature is made using the *terminal private key*, the access to which, like it is to the respective certificates, is performed in compliance with the policy of providing the terminal functioning.

Implementing EAC (version 1.11) the information about algorithm and the supported TA version is stored in `DG14` data group of *`ePassport`* application. Implementing EAC (version 2) the information about algorithm and the supported TA version is stored in `EF.CardAccess`, `EF.CardSecurity`, `EF.ChipSecurity` files of the root *Master File*.

> **ATTENTION!** Another difference in the performance of TA for EAC (versions 1.11 and 2) is that TA version 2 should be performed before CA procedure, but a pair of cryptographic CA keys must already be generated at the time of direct TA performance [24]. In this regard, the SDK introduced a concept of **TA preliminary step** for CA procedure.

4.10. Additional Security Mechanisms

The [24] has a specified number of additional security mechanisms when working with `eID` application:

- **restricted identification** (*RI*);
- **verification** of auxiliary data in the process of TA.

4.10.1. Restricted Identification

The **terminal sector** means some logical area of use of a terminal within the organization of work with electronic documents.

The superior entity (usually – *Document Verifier)* gives to each sector a pair of cryptographic keys. When producing a document for use within a particular sector, the private key is stored in secure memory of the RFID-chip, and the public key is placed in an accessible terminal database.

In the process of restricted identification, the chip reports its unique *sector-specific identifier* in response to the sent sector public key. In compliance with [24] support of work with two different key pairs is possible.

The sector-specific identifier may be used for revocation of electronic document when it is in a special revocation list, which is composed by the superior entity (such as CVCA).

Details about the algorithm and parameters of RI is stored in `EF.CardAccess`, `EF.CardSecurity`, `EF.ChipSecurity` files of the root *Master File*.

RI is available only after passing the preliminary CA and TA procedures.

4.10.2. Auxiliary Data Verification

Working with `eID` application and after successful TA procedure completion there is a possibility of verification of auxiliary data directly by means of the chip without necessity to read the corresponding data groups and performing the verification in the software [24].

Such data include:

- DO's `Community ID`,
- DO's age.

In the first case, a comparison of data transmitted to the chip with contents of `DG18` is executed (byte-to-byte comparison, starting with the first byte until the end of the transferred data), in the second case – a comparison of transmitted date with contents of `DG8` to determine the fact that the document owner was not born later than that date.

4.11. Procedures of Document Authentication

The procedure of document authentication allows:

- performing the effective terminal authorization, by determining the effective type of terminal and its corresponding available set of functionalities for organization of data exchange with the RFID-chip;
- on the basis of the data from the RFID-chip to verify the authenticity of the document;
- using the provided functionality for additional verifications (RI, auxiliary data verification) or service operations (password management, digital signature generation etc.).

4.11.1. Standard Inspection Procedure

This procedure of document authentication (*Standard Inspection Procedure*) is used to confirm the effective type of IS terminal.

It provides access to all data groups of `ePassport` and `eDL` applications, except the sensitive biometric data of fingerprints and iris.

The order for carrying out this procedure is following:

1) for `ePassport` application, by the presence of `EF.CardAccess` and its contents the support of PACE by the RFID-chip as a basic mechanism of SM is determined. In case of such support the secure data access channel is initialized;
2) the application is selected;
3) in case if PACE is not supported, the secure data access channel with BAC/BAP as a basic mechanism is initialized during this step;
4) the first PA phase is performed: `EF.SOD` is read, verification of its digital signature is performed.

In case of a successful step 4 further reading of informational data groups with their integrity verification as part of PA is possible.

4.11.2. Advanced Inspection Procedure

This procedure of document authentication (*Advanced Inspection Procedure*) is used to confirm the effective type of IS terminal.

It provides access to all data groups of `ePassport` and `eDL` applications, including the sensitive biometric data of fingerprints and iris.

The order for carrying out this procedure is following:

1) for `ePassport` application, by the presence of `EF.CardAccess` and its contents the support of PACE by the RFID-chip as a basic mechanism of SM is determined. In case of such support the secure data access channel is initialized;
2) the application is selected;
3) in case if PACE is not supported, the secure data access channel with BAC/BAP as a basic mechanism is initialized during this step;
4) CA procedure (version 1) is performed, that opens a new SM communication channel;
5) the first PA phase is performed: `EF.SOD` is read, verification of its digital signature is performed;
6) TA procedure (version 1) is performed, that opens access to informational groups of sensitive biometric data.

In case of a successful step 5 further reading of informational data groups with their integrity verification as part of PA is possible.

In case of a successful step 6 further reading of information data groups of sensitive biometric data with their integrity verification as part of PA is possible.

4.11.3. General Authentication Procedure

This procedure of document authentication (*General Authentication Procedure*) is used to confirm the effective type of any terminal (depending on the information given by the terminal during the step of procedure initialization).

It provides access to:
- all data groups of `ePassport`, `eDL` and `eID` applications for IS terminal;
- reading (and if provided – updating) of all data groups of `eID` application for AT terminal;
- functions of initialization of `eSign` application (creating of *eSign-PIN* and of a new pair of cryptographic keys for digital signature generation) for AT terminal;
- functions of `eSign` application of generating data digital signature for ST terminal;
- functions of password management for all types of terminal (depending on the used password when initializing SM communication channel).

The general authentication procedure means exclusive use of PACE as a basic SM mechanism and is available only for RFID-chips that support EAC (version 2) [24].

The order for carrying out this procedure is following:
1) by the presence of `EF.CardAccess` and its contents the support of PACE by the RFID-chip as a basic mechanism of SM is determined. In case of such support the secure data access channel is initialized. Otherwise the procedure is unavailable.
2) TA procedure, version 2, is performed.

3) The first PA phase is performed: `EF.CardSecurity` and `EF.ChipSecurity` (if necessary) are read, verification of their digital signature is performed.
4) CA procedure, version 2, is performed, which opens a new SM communication channel.

In case of a successful step 4 further selection of required applications to read informational data groups with their integrity verification as part of PA procedure, as well as using of various functionality of the electronic document is possible.

5. WORKING WITH SDK

5.1. ORGANIZATION OF WORK WITH THE MAIN CONTROL LIBRARY

The main control library SDK **`RFID_SDK.dll`** exports a number of functions, used to work with the RFID-chips readers.

Operation of all functions is organized according to the possible use of *multi-threaded* data processing environment. Call of any functions of the library can be performed by multiple threads of the user application. This, for example, allows organizing data reading from the RFID-chip in the background, leaving the main program interface unlocked.

The library **`RFID_SDK.dll`** developed for the dynamic connection using Windows API **`LoadLibrary()`** function. Pointers to exported functions can be obtained using Windows API **`GetProcAddress()`** function.

After loading the library into memory, it is required to make a call to the initialization function **`_RFID_Initialize()`**.

The main function of the library, through which a user application may initiate all necessary actions to work with RFID-chips, is **`_RFID_ExecuteCommand()`** function. It takes a command triplet as parameters: *command code*, *command input parameter* and *pointer to the container* for the returned results. As any operations of data exchange between the reader and the RFID-chip are time-indivisible (synchronous), the implementation of execution of all commands by the control library is made also by the synchronous scheme. This means that at the time **`_RFID_ExecuteCommand()`** function returns the requested action is fully completed and all the possible results of command execution have been received and are valid.

To receive detailed information about the current actions that occur during the execution of the command, the callback function mechanism is used. Using exported **`_RFID_SetCallbackFunc()`** function it is possible to initialize a pointer to a function (that has **`RFID_NotifyFunc`** type) of user application, that will be called at various stages of the command execution providing the event code (hereinafter - *message*) and additional data in the context of the event. Execution of the callback-function will occur by default in the context of the *inner SDK working threads*. In this case consideration should be given to the ability to ***synchronize*** actions of the main application that uses the library to work with shared resources (data, interface) as well as to *prevent the recursive calls* to **`_RFID_ExecuteCommand()`**, which will be calling it directly from the callback-function.

At the end of the work with the main control library it is required to call **`_RFID_Free()`** function and unload the library from memory using Windows API **`FreeLibrary()`** function.

> **ATTENTION!** **`_RFID_Free()`** must be called from the same thread of the user application as previous **`_RFID_Initialize()`**.

5.2. Additional Data Verification

Starting with the SDK version 2.0, an additional check of compliance with numerous normative documents is performed at all stages of the work:

- for contents of the information and service data groups from the RFID-chip memory;
- for associated resources used during the authentication procedures (certificates for passive authentication, TA certificates and private keys etc.).

The user application will be informed about all ascertained inconsistencies by `RFID_Notification_ISOError` notification message with a numerical code in the message parameter indicating the specific identified situation.

Two types of inconsistencies (remarks) have been defined: *critical* and *non-critical*.

Critical inconsistency shows the impossibility to continue the current operation due to incorrect data being processed (notification codes – **`eLDS_ParsingErrorCodes`** values).

Non-critical inconsistency allows continuing the current operation and leaves the right to the user application to choose a reaction to the given situation (notification codes – **`eLDS_ParsingNotificationCodes`** values).

The user application can choose the *level of strictness* of SDK reaction to the detection of inconsistencies. To define it **`RFID_Command_SetDataProcessingLevel`** command is used, to read the current value – **`RFID_Command_GetDataProcessingLevel`** command.

When choosing a level of strictness of `dplStrictISO` any found inconsistency is critical and its detection interrupts the current operation.

To receive an abbreviation of a notification code (or the return code from SDK function) there is **`RFID_Command_Get_CodeTranscription`** command.

Additionally, there exists a possibility to choose the type of logical data profiler to use with the electronic document in accordance with the requirements of [2] and [3] (default) or [31] by using **`RFID_Command_Set_ProfilerType`** command. **`RFID_Command_Get_ProfilerType`** is used to read the current value. Differences in the use of different types of profilers are in a set of requirements for the structure and content of data of various objects used in the process of working with electronic document. Sets of possible inconsistencies found in the analysis of the data differ respectively.

5.3. RFID-CHIPS READER CONNECTION AND ACTIVATION

«Regula» 7051 RFID-reader can be both a separate device or can be embedded in other devices such document readers «Regula» mod. 7024.xxx. In both cases, when connecting to a free USB-port of PC the OS will determine its presence as a separate device; it will activate the driver and execute the primary initialization. Besides, the reader will be registered in system «*Device Manager*» in «*Smart card readers*» group as «*Regula RF-Reader*».

The main control library SDK supports work with any number of readers, simultaneously connected to the PC, but at one time only one of them can be active.

To determine the total number of readers that are currently connected to the PC there is **`RFID_Command_Get_DeviceCount`** command.

Each device has its own identifier – its corresponding serial number (index) in the general list.

A number of commands allows requesting a particular characteristic of a particular reader in the list by its index:

- **`RFID_Command_Get_DeviceDescription`** – symbolic name of the reader that is determined via system `SCard` service;
- **`RFID_Command_Get_DeviceInstanceID`** – symbolic system identifier of the reader device instance determined with the help of Windows API `SetupDiGetDeviceInstanceID()` function;
- **`RFID_Command_Get_ParentInstanceID`** – symbolic system identifier of the device instance, to which the reader is physically connected (in most cases it is USB Hub), which is determined with the help of Windows API `CM_Get_Device_ID()` function;
- **`RFID_Command_Get_DeviceHardwareID`** – symbolic system identifier of the reader, which is determined with the help of Windows API `SetupDiGetDeviceRegistryProperty()` function;
- **`RFID_Command_Get_DeviceSN`** – reader serial number;
- **`RFID_Command_Get_DeviceDriverVersion`** – version of the device driver.

After initialization of the main control library the device with index **`0`** is activated by default.

All commands for data reading and received messages about the appearance of RFID-chip in the reader scope (or its removal from it) will correspond only to the current active device.

To activate a specific reader there are commands:

- **`RFID_Command_Set_CurrentDevice`** – by reader's index in the list;
- **`RFID_Command_SelectDeviceByName`** – by the symbolic string of the system UID of the parent HUB;
- **`RFID_Command_SelectDeviceBySN`** – by reader's serial number.

Index of the current active device can be received by **RFID_Command_Get_CurrentDevice** command.

Since the functionality of a RFID-reader have extended with the transition to control by PC/SC driver, the implementation of some of them in practice is possible only in conjunction with a specific version of reader's firmware. The same applies to the extension of the functionality of the reader in developing the subsequent SDK versions.

If the version of reader's firmware does not fully implement the functionality of the current SDK version, the user application will automatically be informed of this by **RFID_Notification_Error** message with **RFID_Error_OldFirmware** notification code at the time of reader activation. In this case it is required to update reader's firmware using special utility (see section 1).

To determine the firmware version of the RFID-reader there is **RFID_Command_Get_DeviceFirmwareVersion** command.

If during work with the SDK RFID-reader was physically unplugged from the PC, the user application will be informed by **RFID_Notification_PCSC_ReaderDisconnected** message.

When connecting to the PC a new RFID-reader working under PC/SC driver control, the user application will be informed by **RFID_Notification_PCSC_ReaderListChanging** message in the beginning of the procedure of rebuilding the list of available RFID-readers and **RFID_Notification_PCSC_ReaderListChanged** at its end.

5.4. Reader Parameters

5.4.1. RFID-chip Detection Modes

After initialization of the main control library and activation of the RFID-reader the user application receives a possibility to track the moments of RFID-chip appearance in the scope of the reader antenna and its removal from it. In both cases, a call to **RFID_NotifyFunc** callback-function, preinstalled with **_RFID_SetCallbackFunc()**, will be carried out. **RFID_Notification_DocumentReady** notification code and its logical value: `true`, if the RFID-chip has appeared in sight, or `false`, if it was removed from the scope, will be transferred to **RFID_NotifyFunc** as parameters.

Two modes are provided in the SDK:

- automatic;
- manual.

In the first case the search of chip will be carried out automatically. In the second case – only after execution of **RFID_Command_DocumentDone** command with `ddmDetectChip` parameter.

A variant of the used detection mode is set by the input parameter of **_RFID_Initialize()** function (`CtrlRF_Auto` or `CtrlRF_Manual`) or by **RFID_Command_Set_DetectionMode** command.

To determine the current status of chip presence in the scope of the reader there is **RFID_Command_IsDocument** command.

5.4.2. Mode to Ignore RFID-chips Supporting only Protocol ISO/IEC 14443-3 (MIFARE® Classic Protocol)

To connect the SDK main control library in the mode of ignorance for RFID-chips, which are not supporting the protocol ISO/IEC 14443-4, it is required to use the value `CtrlRF_14443_4_Only` when forming the parameter of its initialization function **_RFID_Initialize()**. In this case, the presence of such chips in the field of the reader antenna will not be detected.

5.4.3. Data Exchange Speed between the Reader and the RFID-chip

To restrict the maximum allowed data exchange speed between the reader and the RFID-chip there is **RFID_Command_Set_OperationalBaudRate** command.

5.4.4. Size of Operating Data Buffer for Reading

A possibility of implementing data exchange between the reader and the RFID-chip is provided by the standard [20] using the commands of extended length (`extended Le`), which gives a large benefit for the total time of data reading.

Using the commands of single length, the data exchange between the reader and the chip is carried out in portions not exceeding 256 bytes. Using the commands of extended length, the buffer size is limited to 64 kilobytes, which in case of support of such working mode by the chip allows reading the data group contents with significantly less number of queries.

To specify the desired size of the buffer for read data **`RFID_Command_SetTransferBufferSize`** command is used.

When a parameter for this command is set to `0` the extended length commands to read data will not be applied.

When a parameter for this command is set to `-1` an attempt will be made to use the extended length commands for reading using buffer size equal to the size of the retrieved file.

When a parameter for this command is set to any other value attempts of using the extended length commands will be made if the given value is more than **256**.

Since the support of extended length commands is optional and may be implemented not by all chips, the evaluation of this characteristic is carried out by the results of the first attempt of using such command. If the result of such command is unsatisfactory (the chip returned an error code, incorrect or incomplete data have been read), the data reading mode for this specific chip will be automatically transferred to using the commands of single length.

If the chip has support of extended length reading commands, it will be indicated by **`RFID_Notification_PCSC_ExtLengthSupport`** notification message with a corresponding value of the notification parameter (`true` or `false`).

To switch to the use of single length commands it may be necessary to reinitialize the RFID-chip completely – to finish the current work session with a document and to open a new one with preliminary correction of the reading data buffer size. In this case, the current reading operation will be aborted with **`RFID_Error_PCSC_ExtLe_Failed`** return code from **`_RFID_ExecuteCommand()`**.

5.4.5. Antenna Parameters

Removed since version SDK 3.5.

5.4.6. Completion of Work with RFID-chip

`RFID_Command_DocumentDone` command serves for completion of each communication session with the RFID-chip.

One of **`eRFID_ManualChipDetectionMode`** values serves as the command parameter, which sets the search mode for a new RFID-chip in the scope of the reader when working in the mode of *manual* detection:

- `ddmChipPowerOff` – switch off the power without the search for a new chip;
- `ddmDetectChip` – search for the first chip among all those present in the scope of the reader (with preliminary full disconnection of the field – the removal of power for all present chips);
- `ddmDummy` – no actions to be carried out.

5.5. SDK Parameters

5.5.1. Logging

To enable/disable SDK logging there is **RFID_Command_BuildLog** command.

The operation log is a text file that records the sequence and all the results of intermediate actions performed during the execution of SDK commands, including the contents of data exchanged between the reader and the chip during the transmission of information between them.

RFID_Command_UseDeviceDriverLog command is used to include information about the operations performed at the device driver/firmware level in the generated work protocol.

The operation log begins to form after its activation and is constantly written to disk under the name of `RFID.log` to the directory set by **RFID_Command_LogDirectory** command, or by default:

```
<local_user_profile>\AppData\Local\Regula\Debug\.
```

For immediate recording of the current contents of the log in the file with the specified name there is **RFID_Command_FlushLog** command. After its execution formation of the operation log begins anew.

5.5.2. The Parameters of the Passive Authentication

To verify the digital signature of document and master list security objects when performing passive authentication search for the corresponding certificates and check of their validity is carried out (see section 4.8.1).

As a source of certificates serves the local PKD copy – a database that is a set of binary files of certificates, certificate revocation lists (*CRL*) [6] and master lists [34]. LDIF format [32] can be used to represent the contents of master lists and ICAO PKD [33].

The path to the directory containing a set of PKD files for passive authentication is specified by **RFID_Command_Set_PassivePKD** command. The current value of this SDK parameter is requested by **RFID_Command_Get_PassivePKD** command. The directory defined this way may contain any number of nested directories. Search for the required certificates for the particular electronic document is carried out automatically.

Because both the PKD and the security object itself may serve as a source of DS-/MLS-certificate for verification of digital signature of the security object, the priority of using a certificate from a particular source is set by **RFID_Command_Set_DS_Cert_Priority** command. The current value of this SDK parameter is requested by

RFID_Command_Get_DS_Cert_Priority command. The parameter value **0** of this command sets a priority of SO, **1** – PKD.

In both cases, the search for DS-/MLS-certificate will be performed first in PKD. If it is not found there and priority of SO is set, DS-certificate from SO will be selected to perform PA. Otherwise search for DS-/MLS-certificate will be completed with setting an appropriate error code.

To verify the digital signature of DS-/MLS-certificate CSCA-certificate is required. Though for final decision-making on the success of such a check not only its formal passage is required, but also a certain *level of trust* to the source of the used CSCA-certificate. Since this issue is the subject of policy for each individual terminal, the level of such trust can be determined exclusively by the user software.

To set the necessary level of trust for the CSCA-certificates contained in the local PKD there is **RFID_Command_Set_TrustedPKD** command. The current value of this SDK parameter is requested by **RFID_Command_Get_TrustedPKD** command.

In case of absence of the required level of trust to the used CSCA-certificate the user application will be informed by **RFID_Notification_ISOError** notification message with `ntfLDS_SOD_Signer_DSCert_RootIsNotTrusted` code.

In a case of SO_{ML} digital signature verification, corresponding CSCA-certificate can be included in SO_{ML} data structure itself. It is possible to limit the use of CSCA-certificates submitted by individual data files only with **RFID_Command_Set_UseExternalCSCA** command. The current value of this parameter is requested by **RFID_Command_Get_UseExternalCSCA** command.

When determining inconsistencies associated with SO_{ML} digital signature verification, all certificates *belonging to **contents** of the master list* will be marked correspondingly and, in the case of further use to verify SO_{D} digital signature, user software will be informed by **RFID_Notification_ISOError** notification messages with following codes

```
ntfLDS_Auth_MLSignerInfo_Certificate_Validity
ntfLDS_Auth_MLSignerInfo_Certificate_RootIsNotTrusted
ntfLDS_Auth_MLSignerInfo_Certificate_CantFindCSCA
ntfLDS_Auth_MLSignerInfo_Certificate_Revoked
ntfLDS_Auth_MLSignerInfo_Certificate_SignatureInvalid
```

5.5.3. Definition of the Local Public Key Certificates Library for Terminal Authentication

To perform TA, the access to some additional resources is required: to TA-certificates for a specific document and the corresponding private cryptographic key (see sections 4.7, 4.9.3). These resources are presented as a set of files.

CV-certificates are represented as files with the extension "**.cvcert**", containing binary TLV-certificate data [24, part 3, §§ C.1, D]. The corresponding private key file must be coincident with the certificate file name with the extension "**.pkcs8**" and contain binary key data in the format specified in [10].

The path to the directory containing a set of files for TA is defined by **`RFID_Command_Set_EAC_PKD`** command. The current value of this SDK parameter is requested by **`RFID_Command_Get_EAC_PKD`** command. The directory defined this way may contain any number of nested directories with a set of resources for various documents. Search for the required elements among all available data for a specific electronic document is carried out automatically.

5.6. ORGANIZATION OF WORK WITH ELECTRONIC DOCUMENT

5.6.1. Modes of Operation

Version 3.1 SDK provides three ways of working with the electronic document in the following modes:

- batch;
- session;
- scenario.

The **batch operation mode** allows automatic executing of all necessary authentication procedures and data reading from the RFID-chip memory based on predefined set of actions and the provision of all required additional information (certificate chain for TA, data access password for organization of SM-channel etc.).

The results of data reading in the batch mode become available only after the completion of the reading command – after the return from the corresponding **`_RFID_ExecuteCommand()`** call.

This operation mode is fully consistent with SDK of the previous versions and is available only for reading and authentication of electronic documents data containing *`ePassport`* application [1], [2], [3].

The **session operation mode** provides maximum flexibility in organizing a communication session with the electronic document with all supported types of applications (*`ePassport`*, *`eID`*, *`eSign`*, *`eDL`*). It is based on the principle of organizing the document working session. After session opening there is a possibility of an independent execution of individual operations (such as selection of the required application, reading of definite data groups, authentication of the specified type, etc.) for implementation of the necessary working scenario with the document. In this case all the results of carrying out operation immediately registered in the data object of the current session, always accessible to the calling software.

The **scenario operation mode** is an adaptation of the batch mode for the session work with electronic documents. It also provides automatic execution of all necessary authentication and data reading procedures based on a predetermined set of actions. All additional information is required immediately prior to an action by the callback function.

Besides, in the scenario mode there is an opportunity to select not only a variant of CA or TA procedure performance, but of PACE procedure as well.

Set of the necessary actions (scenario) is specified as XML-structure input parameter to the corresponding **`_RFID_ExecuteCommand()`** call, as well as input and output data that is transferred via callback function parameter.

In the scenario mode, as in the batch mode, data readout result is accessible only after the command execution.

5.6.2. Representation of Read Data

Data read from the RFID-chip can be represented as follows:

- as a list of binary data arrays, which are an exact copy of the data stored in the memory of RFID-chip, without any additional formatting (**TDocBinaryInfo** with list elements of **TBinaryData** type);
- as a list of structures, corresponding to different types of data. They do not contain service information used when formatting to record in the chip memory (**TDocBinaryInfo** with list elements of **TBinaryData** type);
- as a list of binary data arrays, which are an exact copy of the graphic files stored in the memory of RFID-chip (**TOriginalRFIDGraphicsInfo** with list elements of **TOriginalRFIDGraphics** type);
- as a list of logically selected document filling fields that contain text and graphics information (**TDocVisualExtendedInfo** with list elements of **TDocVisualExtendedField** type, **TDocGraphicsInfo** with list elements of **TDocGraphicField** type).

Each of the lists is represented by **TResultContainer** object. Type of the data representation is defined by **eRFID_ResultType** values (`result_type` field contents).

Logical data type of the elements of **TDocVisualExtendedField**, **TDocGraphicField**, **TBinaryData**, **TOriginalRFIDGraphics** is defined by the field `FieldType`, which may contain one of **eVisualFieldType**, **eRFID_VisualFieldType**, **eGraphicFieldType** or **eRFID_DataFile_Type** values.

The values **eVisualFieldType** and **eRFID_VisualFieldType** are used for designation of **TDocVisualExtendedInfo** elements, **eGraphicFieldType** – for **TDocGraphicsInfo** and **TOriginalRFIDGraphics** elements, **eRFID_DataFile_Type** – for **TDocBinaryInfo** elements.

It should be noted that for storage and transmission of data read by **RFID_Command_ReadProtocol3** command, only the structure corresponding to **RFDP_Raw** type is used. When performing sequential data reading by **RFID_Command_ReadProtocol3** and **RFID_Command_ReadProtocol4** commands, this structure stores the results of both commands in a merged list.

5.6.3. Data Reading Result Acquisition

Access to the results of the data read operation is performed as follows:

- in the ***batch mode*** the return value from **_RFID_ExecuteCommand()** function for **RFID_Command_ReadProtocol3/RFID_Command_ReadProtocol4** commands is a pointer to **TResultContainerList** list object, containing elements for all data representation types (see section 5.6.2);

- in the ***session mode*** the results of the ongoing operations are available all the time – immediately after certain actions – through the pointer to the working **TRFID_Session** object (see section 5.8.2);

- in the ***scenario mode*** the return value from **_RFID_ExecuteCommand()** function for **RFID_Command_Scenario_Process** command is `VARIANT *`, which will point to XML-representation of **TRFID_Session** structure containing results of the performed data reading session (see section 5.9.3).

Furthermore, after the read operation in all modes of operation there is an additional method of access to the results – with **_RFID_CheckResult()** and **_RFID_CheckResultFromList()** functions, providing access to a copy of the data.

This mechanism is applicable for reception of *XML-representation* of the results as well.

As the input parameters **_RFID_CheckResult()** in `type` parameter a type of requested data is accepted (one of **eRFID_ResultType** values), in `output` parameter – format of returned data (one of **eOutputFormat** values) and in `param` – parameters of data transfer (in the context of the value `output`).

If the value returned from **_RFID_CheckResult()** *is less than* `0`, it is one of **eRFID_ResultStatus** error codes.

If the value returned from **_RFID_CheckResult()** *is larger than* `0`, it is actually a pointer to **TResultContainer** structure, containing the requested data. It may be used directly, casting to **TResultContainer** `*` type, or for access to the data fields contents separately, with the help of **_RFID_CheckResultFromList()** function (when requesting acquisition of the result **RFID_ResultType_RFID_TextData** or **RFID_ResultType_RFID_ImageData**). In the second case a call of **_RFID_CheckResult()** function is an intermediate step for specific data acquisition.

As the input parameters, **_RFID_CheckResultFromList()** accepts the descriptor of results list, received after **_RFID_CheckResult()** call (`container` function parameter), identifier of data transfer mechanism (one of **eOutputFormatField** values in `output` function parameter) and parameters of data transfer (`param`, in a context of `output` parameter value). The value returned by the function contains a code of the transferred field (one of **eVisualFieldType**, **eRFID_VisualFieldType** or **eGraphicFieldType** values). In case of error or when the end of the field list is reached one of **eRFID_ResultStatus** values is returned.

Two ways of the transferring of result list fields contents are provided (is given in `output` function parameter):

- through Windows clipboard (for text fields from **TDocVisualExtendedInfo** structure and graphic images from **TDocGraphicsInfo** structure);
- through a file (for graphic images from **TDocGraphicsInfo** structure).

In the first case `param` function parameter must contain a window handle (`HWND`), with which the clipboard will be connected. In the second case – a pointer to the character string containing the file name, which the image will be saved under. Graphic encoding file format is selected on the basis of the file name extension. Commands **RFID_Command_Set_Graphics_CompressionRatio** and **RFID_Command_Get_Graphics_CompressionRatio** are used to set and read the level of compression using the corresponding image recording formats (e.g., JPG).

A full list of graphic file formats extensions, available for use, can be received by **RFID_Command_Get_AvailableGraphicFormats** command. A character string consisting of three-letter graphic files extensions, separated by a symbol «;», for example, «BMP; JPG; TIF», is returned to the user application.

Thus, for acquisition of the contents of all text or graphic fields contained in the resulting structures by the request to formation of **RFDP_FullyParsed** result, the following actions are required:

1) call the **_RFID_CheckResult()** passing **RFID_ResultType_RFID_TextData** or **RFID_ResultType_RFID_ImageData**;
2) using the received descriptor, call the **_RFID_CheckResultFromList()** until the moment of reaching the list end (`RFID_ResultStatus_EndOfList` return code).

After return from **_RFID_CheckResultFromList()** the requested data are either located in the clipboard or saved in the file with the given name.

Providing one of the values `ofClipboard_XML`, `ofFile_XML` or `ofXML` in the parameter `type` of **_RFID_CheckResult()** function, formation of XML-representation of requested type data structure will be performed. `XML_buffer` field of the returned **TResultContainer** structure will be initialized by the respective pointer to the symbol array. The size of the array will be specified in `XML_length` field of the same structure.

Giving **ofClipboard_XML** the symbol array of XML result representation will be in addition stored in Windows clipboard, connected with the window, the handle of which (`HWND`) has been specified in `output` parameter.

Giving **ofFile_XML** the symbol array of XML result representation will be in addition stored in a text file. The pointer to the character string containing the file name must be specified in the parameter `output`.

5.7. Batch Operation Mode

5.7.1. Determination of RFID-chip Characteristics

After receiving of **RFID_Notification_DocumentReady** message the user application can initiate the procedure of data reading from the RFID-chip memory. However, taking into consideration the differences in the structure of data and parameters between different types of RFID-chips (see section 4.1), it is required at first to determine the specific chip type and its main characteristics, such as the amount of memory, supported communication protocols, the rate of data transmission/reception etc.

To receive information about the characteristic of the RFID-chip, located within sight of the reader, there are commands:

- **RFID_Command_Get_ReadCardProperties**
- **RFID_Command_ReadCardPropertiesExt**
- **RFID_Command_ReadCardPropertiesExt2**
  (for readers with firmware version 21.00 and higher)

As a result of its execution user application receives a pointer to the corresponding data structure:

- **TRFCardProp**
- **TRFID_CardPropertiesExt**
- **TRFChipProperties**

containing information about chip's main characteristics.

The user application must build an algorithm of its work with electronic document based on these data.

ATTENTION! Each time **RFID_Command_Get_ReadCardProperties** command executed all data obtained during the previous reading are destroyed when reallocating memory for the new result.

5.7.2. Data Reading via MIFARE® Classic Protocol

To read data from a chip that supports ISO/IEC 14443-3 (MIFARE® Classic Protocol) standard communication protocol there is **RFID_Command_ReadProtocol3** command. It is available only for chips, `TRFCardProp` characteristic structure of which contains the value `true` in the `Support_Mifare` field (see section 5.7.1).

Reading data by **RFID_Command_ReadProtocol3** command two elements in the list of results are formed (in **TDocBinaryInfo** structure – for the type of **RFDP_Raw** data representation):

- *read data*;
- *an array of flags of correctness of reading of each chip memory sector*.

The type of these list elements is specified in `FieldType` field of **TBinaryData** structure, which is the basic type for **TDocBinaryInfo** elements. It will contain `dftMIFARE_Data` value for data, `dftMIFARE_Validity` – for flags of reading correctness.

For example, if the amount of chip memory 1 Kb, `dftMIFARE_Data` array will contain 1024 bytes of data, and `dftMIFARE_Validity` array – 16 bytes with the value `0`, if the respective sector was read with an error, or `1`, if the data were read successfully.

During execution of **RFID_Command_ReadProtocol3** command the user application will receive **RFID_Notification_ReadProtocol3** message, notifying about the beginning and end of data reading operation.

Since the data recorded in the memory of such RFID-chips do not have standardized logical structure, their *logical* analysis by the SDK means is not performed and is prerogative of the user application.

5.7.3. Authentication using MIFARE® Classic Protocol

The procedure of authentication for each sector of chip memory must precede any operation of data reading from the RFID-chip via protocol MIFARE® Classic Protocol.

Two 6-byte sequences – `KeyA` and `KeyB` – are used as the sector authentication keys.

Three authentication modes are provided by the SDK:

- by default – when the bytes sequences '`0xFF 0xFF 0xFF 0xFF 0xFF 0xFF`' for both keys are used as the keys for all sectors '`0xFF 0xFF 0xFF 0xFF 0xFF 0xFF`' (**mkmDefault**);
- using a single key for all memory sectors (**mkmSingleKey**);
- using a separate key for each of the memory sectors (`mkmFullKeyTable`).

By **RFID_Command_SetMIFARE_KeyMode** command the authentication mode type is set specifying one of **eMIFARE_KeyMode** constants. By **RFID_Command_GetMIFARE_KeyMode** command the current value of authentication mode is requested. By **RFID_Command_SetMIFARE_KeyTable** command the values of authentication keys are assigned. **TMIFARE_KeyTable** structure is given as the parameter of this command, containing two arrays of 40 six-byte keys (`A` and `B`), which ensures covering of 40 memory sectors (up to 4 Kb). *The first elements* of these arrays are used in `mkmSingleKey` authentication mode as a single key. By **`RFID_Command_GetMIFARE_KeyTable`** command the current values of authentication keys are requested.

> **ATTENTION!** Considering the peculiarities of the PC/SC-driver reader, starting with the SDK version 3, support of the **default** authentication mode only is implemented. The user application cannot influence this procedure.

5.7.4. Data Reading via ISO/IEC 14443-4 Protocol

To read data from the memory of RFID-chips that support the protocol ISO/IEC 14443-4, there is **`RFID_Command_ReadProtocol4`** command.

The user application must pass a combination of **`eRFID_DataGroups`** flags defining a set of read data groups as a parameter of this command (see section 4.3.1).

The beginning of data reading operation and its end is marked by sending **`RFID_Notification_ReadProtocol4`** message.

At the first stage of executing **`RFID_Command_ReadProtocol4`** command the application *`ePassport`* is selected and the operation of `EF.COM` service group reading is performed to determine the set of present informational data groups.

The contents of `EF.COM` service data group will be stored in in the returned list of results (the element with `dftPassport_COM` type in **`TDocBinaryInfo`** structure for **`RFDP_BinaryParsed`** type of data representation).

If the command parameter (set of read groups) was given a value of **`0`** (**`NULL`**), there will be return to the calling function of the user application after processing of `EF.COM`. If, however, was given a non-zero data group combination, execution of the command **`RFID_Command_ReadProtocol4`** will continue.

Reading of any informational data group will be performed if it is present in the chip memory, and if it was specified in the command parameter. Otherwise, the element with the appropriate data type will be missing in the generated list of results.

The beginning of the reading of any informational data group and its ending is marked sending **`RFID_Notification_PCSC_ReadingDatagroup`** message. The code of read file from the **`eRFID_DataFile_Type`** is implemented (contained in the low order `WORD`) in the code of this notification.

If a situation arises when **`RFDP_FullyParsed`** was chosen as one of the required types of read data representation, but the main control library cannot recognize the encoding format of a graphic image, the user application receives announcement through **ntfLDS_UnsupportedImageFormat** notification. However, this does not affect the formation of results of other types.

A progress of the data reading command execution is marked with sending of **RFID_Notification_Progress** message. The numeric value that is passed with this message defines the amount of read data in percent of the total data amount of the requested information groups.

The **RFID_Command_CancelReading** command serves for interruption of the current reading operation.

In case of a situation when the requested file is missing, the user application will be informed by **RFID_Notification_PCSC_FileNotFound** notification message with a file code in the low order `WORD`.

In case of reaching the end of the file prior to receipt of the requested data amount the user application will be informed by **RFID_Notification_PCSC_EndOfFile** notification message with a file code in the low order `WORD`.

In case of absence of access rights to the requested file for the terminal the user application will be informed by **RFID_Notification_PCSC_FileAccessDenied** notification message with a file code in the low order `WORD`.

The SDK provides possibility to read non-standard files, use of which is not provided by specifications [2] and [3].

To identify such files when working in the batch mode there is **RFID_Command_SetUserDefinedFilesToRead** command. It uses a pointer to **TRFID_FilesList** list object as a parameter, which may contain description of up to 32 different files.

To include files from this list to the common **RFID_Notification_ReadProtocol4** reading operation it is required to add `RFDG_USER` flag to the set of read groups.

Giving the corresponding element of the list of files with `fidtLocal_Path` value in `id_type` field, for that file only the operation of selection without reading its contents will be executed.

5.7.5. Protected Data Reading (BAC)

In case if the access to data in memory of the RFID-chip is protected using the BAC mechanism [3], it is necessary to present a special key sequence to read them, and this is the text of the document MRZ previously read using the procedure of OCR.

The fact that data is protected is determined when attempting to read EF.COM. In this case **_RFID_ExecuteCommand()** will stop execution of **RFID_Command_ReadProtocol4** command, notify the user application by **RFID_Notification_SM_Required** message and return RFID_LAYER6_SECURITY_MANAGER code. The user application must set the data access key for the current document and repeatedly execute **RFID_Command_ReadProtocol4** command.

To define the data access key there is **RFID_Command_SetCryptKey** command, to acquire the current key value – **RFID_Command_GetCryptKey**.

The user application must present the full text of document MRZ in the array of 128 characters. The recognized strings of MRZ must be placed starting from the very first array element, one after another, without additional separators. For example, for documents of ID-3 type, MRZ of which consists of two strings of 44 characters, the text of the first string must be located from the 0 to the 43rd array element, the text of the second string – from the 44th to the 87th. The contents of the remaining array elements are ignored.

ATTENTION! When working in the batch mode use of CAN and SAI passwords for data reading organization is not provided.

The user application is informed about the result of SM-channel opening (true or false) by **RFID_Notification_SM_Established** notification message.

5.7.6. Protected Data Reading (EAC)

If access to the data in the RFID-chip memory is protected using EAC mechanism it is required to execute a number of additional procedures to read them.

According to [1] EAC mechanism may be used to restrict access to contents of informational data groups that contain biometric information on fingerprints (DG3) and iris (DG4) of the DO and is used as a supplement to the mandatory implementation of BAC protection mechanism.

The very presence of EAC protection is defined by the presence of DG14 among the informational data groups, which contains information about the used cryptographic algorithms and keys.

ATTENTION! If DG14 data group was not included in the set of read data groups when executing **RFID_Command_ReadProtocol4** command, the procedure of opening access to EAC-protected data groups will not be performed and the results of their reading will not be included in the total result.

To provide access to protected data groups, EAC-protection mechanism provides execution of two additional authentication procedures: *Chip Authentication* (*CA*) and *Terminal Authentication* (*TA*).

ATTENTION! When working in the batch mode SDK supports only realization of EAC protocol (version 1.11).

CA is performed automatically – immediately after the successful setting of BAC access key and successful data reading from `DG14`.

TA is performed later – after successful CA and after data reading of other information groups, which are not protected by the EAC mechanism.

With the success of TA (if all the required additional resources exist and are valid), access to protected data groups opened and their contents are read ordinary.

The user application will be informed about the beginning of any authentication procedure by **`RFID_Notification_ACProcedure_Start`** message with one of **`eRFID_AccessControl_ProcedureType`** constants in the low order `WORD`. **`RFID_Notification_ACProcedure_Finish`** message informs about the procedure completion and contains the status code of procedure execution (the corresponding value from **`eRFID_ErrorCodes`**) as a parameter.

ATTENTION! If none of EAC-protected data groups was included in the set of read data groups when executing **`RFID_Command_ReadProtocol4`** command, TA procedure will not be performed.

5.7.7. Passive and Active Authentication

To confirm the authenticity of data read from the memory of travel document RFID-chip, as well as the belonging of the chip exactly to this document, a possibility is provided to perform procedures of document passive and active authentication [3].

Passive authentication is to verify the integrity of data of information groups being read by comparing the results of the comparison of hash-functions stored in `EF.SOD` service data group at the stage of document personification, and the values computed over the data acquired directly from the chip. In case of mismatch between these values for any information group one can say that the data have undergone a modification.

Besides, check of the digital signature of `EF.SOD` document security object is performed.

The procedure of passive authentication is performed when executing **RFID_Command_ReadProtocol4** command automatically in case if RFID-chip memory has EF.SOD service data group.

Initial data from EF.SOD will be put in the returning list of results as elements with dftPassport_SOD type in **TDocBinaryInfo** structure for RFID_ResultType_RFID_RawData type of data representation (see section 5.6.2).

The beginning of the digital signature verification procedure for EF.SOD is accompanied by sending of **RFID_Notification_PA_Request** notification message. A pointer to **TPassiveAuthenticationData** structure is passed as the message parameter, containing description of DS-certificate with the public key required for signature verification. The need to use one of the variants of the public key locating (for Issuer and serial-Number or subjectKeyIdentifier, see section 4.8.1) for the digital signature object is determined by the contents of the respective fields in **TPassiveAuthenticationData** structure.

The user application may either ignore this message or provide its own certificate by initializing the appropriate fields in the **TPassiveAuthenticationData**.

The beginning and the end of the procedure of formation of the corresponding certificate chain (see section 4.8.1) is accompanied by sending of **RFID_Notification_PA_CertificateChain** notification message.

Active authentication is to verify the origin of the data stored in the memory of the RFID-chip, in order to detect the fact of its reprogramming by the data from another document (cloning).

The procedure of active authentication is also performed when executing **RFID_Command_ReadProtocol4** command automatically in case if the RFID-chip memory contains EF.DG15 informational data group.

The results of passive and active authentication are stored in **TRF_Authentication** structure, which is put in the list of results as the element with dftAuthenticityV2 type in **TDocBinaryInfo** structure for RFID_ResultType_RFID_BinaryData data representation type (see section 5.6.2).

ATTENTION! If EF.SOD or DG14 were not included in the set of reading data groups when executing **RFID_Command_ReadProtocol4** command, the corresponding authentication procedure will not be performed, and the results of its performance will not be included in the total result.

5.7.8. Data Reading Procedure Completion

Once the user application has received the results of data reading and performed their processing, it must execute the command of reading procedure completion **RFID_Command_DocumentDone**.

To ensure data integrity for their subsequent use it is recommended to create a copy of the acquired results in the address space of the application, which performs MCL function call.

At the end of the batch reading operation some summary information about the performed procedure is sent to the user application by a set of notification messages (Table 2).

Message	Data (**val** contents)
RFID_Notification_PCSC_BytesReceived	The total amount of data received from the RFID-chip with respect to all service information, bytes
RFID_Notification_PCSC_TotalReadingTime	The total data reading time, ms
RFID_Notification_PCSC_DataReceived	The total amount of information and service groups data received from the RFID-chip, bytes
RFID_Notification_PCSC_BytesSent	The total amount of data transmitted to the RFID-chip, bytes
RFID_Notification_PCSC_TotalReadingSpeed	The average reading speed, kB/s · 1000
RFID_Notification_PCSC_TotalProcessTime	The total run time of reading procedure, ms

5.8. Session Operation Mode

5.8.1. Management of Document Working Session

After receiving **RFID_Notification_DocumentReady** message the user application can initiate the procedure of data reading from the RFID-chip memory or attempt to use any of its functionality as a part of created document working session.

To open the session there is **RFID_Command_Session_Open** command. A pointer to **TRFID_Session** object serves as the result of execution of this command, which will serve as the operating object of the session during all time period of its activity (until the closing by **RFID_Command_Session_Close** command).

All commands of work with session (session commands) require use of **TRFID_Session *** received thus as the one of their input parameter, making their use available only in the context of the current active session.

After closing of the session **TRFID_Session** contents will be available until the opening of the new session of work with the document or until the executing of **RFID_Command_ClearResults** command.

5.8.2. Access to the Results of the Session

The results of execution of a session command is immediately registered in the respective **TRFID_Session** object, which makes it possible for the user software to fully control the process of interaction with the electronic document.

Results of referencing the resources of various applications of the electronic document are registered in `pApplications` list: the contents of read files, the results of data analysis and their logical parsing, etc. The elements of this list are pointers to **TRFID_Application** objects.

To store the results of file processing of the root *Master File* there is `pRootFiles` list. The elements of this list are pointers to **TRFID_DataFile** objects.

`pAccessControls` list is used for registration of the results of the various procedures of authentication and secure data access. The elements of this list are pointers to **TRFID_AccessControlInfo** objects. Available variants for a particular procedure, formed on the basis of the read data (for example, variants of CA procedure based on `EF.CardAccess` and `EF.CardSecurity` service files reading), are registered in the corresponding list element (`TRFID_AccessControlInfo.pOptions`). The result of the procedure is registered in `TRFID_AccessControlInfo.Status` field.

Information on document security objects found in the currently read data is registered in pSecurityObjects list. The elements of this list are pointers to **TRFID_SecurityObject** objects.

Status field corresponds to the result of execution of the last session command and may contain one of **eRFID_ErrorCodes** constants.

A more detailed description of assignment of fields of **TRFID_Session** structure is given in the section 6.3.66.

In addition to the above working session object direct access, the contents of the corresponding **TRFID_Session** object will be stored in the list of results as the element with dftSession type in **TDocBinaryInfo** structure for RFID_ResultType_RFID_BinaryData data representation type (see section 5.6.2).

5.8.3. Opening of the Session and Determination of Basic Functionality of the Electronic Document

When performing the command of session opening **RFID_Command_Session_Open** the following actions are performed automatically:

- determination of the characteristic of electronic document RFID-chip;
- selection of the root *Master File*;
- attempt to read EF.CardAccess service file to determine:
  1) PACE support as a basic mechanism of SM communication channel organization and the parameters of the procedure;
  2) version of supported EAC procedures (CA, TA).

In the absence of EF.CardAccess or if there are any reading errors the further work with a document is limited to the use of *ePassport* and *eDL* applications and EAC (version 1.11) only, for which the presence of that service file is not required.

Characteristic of the RFID-chip of the electronic document is registered in CardProperties field of **TRFID_Session** structure (see section 5.7.1). In addition, for readers with firmware version 21.00 and higher, the information structure **TRFChipProperties** is registered in pRootFiles list as an additional element with the dftChipProperties type (see section 5.8.10, the contents of the structure are in the FileData field of this element).

pAccessControls list is filled with elements corresponding to all supported types of procedures of authentication and secure data access: BAC/BAP, PACE, CA, TA, AA, and RI.

ATTENTION! Either in the absence of `EF.CardAccess` or if any critical data contents analysis error appeared, the list element corresponding to PACE will not be included in `pAccessControls` list.

In case of `EF.CardAccess` presence the following are determined in accordance with [23] and [24]:

- available variants of PACE performance (version, algorithms);
- available variants of CA performance (version, algorithms);
- TA support (version).

This information may be updated later, based on reading other service files:

- `EF.CardSecurity` / `EF.ChipSecurity` (for CA, TA, RI);
- `EF.DG14` from application *`ePassport`* (CA, TA version 1);
- `EF.DG15` from application *`ePassport`* (AA);
- `EF.CVCA` from application *`ePassport`* (TA version 1).

This is done automatically during the execution of **`RFID_Command_Session_ReadFile`** command for the respective files.

In addition, for TA version 2, specification of the information about the available variants takes place after successful completion of PACE procedure, as the chip responds with the identifier of the working CVCA-key at that point.

Information on variants for a particular procedure is stored in `TRFID_AccessControlInfo.pOptions` list of the corresponding element of `pAccessControls`.

Pointers to **`TRFID_AccessControl_Option`** objects are elements of this list, each of which describes one available variant of procedure performance in the definite fields:

- `Version` – procedure version (for PACE, CA, TA);
- `Scheme` – algorithm of the used cryptographic scheme (CA, TA protocol) or TA public key algorithm (is specified after the procedure itself); for PACE procedure contains text identifier (OID) of the used public key standardized domain parameters;
- `KeyAlgorithm` – public key algorithm (for PACE, CA, AA, RI) or working CVCA-key identifier (for TA);
- `ChipIndividual` – a sign of key usage availability for privileged terminals only (for CA) [24, part 3, A1.1.7, A.1.2].

Note. A specification of the algorithm of applicable cryptographic scheme for TA occurs directly during the procedure itself, because it depends on the contents of the certificate used by the terminal.

The presence of BAC support is considered constant for all electronic documents.

Actual requirement for BAC/BAP performance is determined during the process of reading of any file that requires the organization of SM channel (RFID_LAYER6_SECURITY_MANAGER return code for the reading command), and when there is no PACE support of the electronic document.

After session initialization and prior to determination of a particular procedure support the Status field of the respective **TRFID_AccessControlInfo** objects will contain the **RFID_Error_NotAvailable** value, after the time of support determination and to the moment of the procedure – the **RFID_Error_NotPerformed** value.

In a case of CardInfoLocator data location in EF.CardAccess [24, part 3, A.1.1.5] an individual data item with acptCardInfo procedure type is added into pAccessControls list. Its list of options consists of a single element, whose Scheme field contains URL string, KeyAlgorithm – fid data, and ChipIndividual – sfid data of CardInfoLocator informational structure. This procedure type is for information purposes only and its use for a real authentication procedure is ignored (see section 5.8.7).

5.8.4. Setting Terminal Configuration

The first *mandatory* operation after the opening of electronic document working session is the operation of **definition of terminal configuration**.

This operation allows defining (or limiting) *a set of informational* and *functional capabilities, delegated by the electronic document and SDK to the current user* (user software), by the **declared** information on the terminal.

Restrictions on the use of certain capability by the terminal are imposed by the respective specifications: [1], [23], [24], [25]. These restrictions are identified both on the logical level of SDK operation (for example, right of using the passwords of different types for a particular terminal), and in the process of direct work with the document (in the effective terminal authorization, see section 4.7, Table 1).

Definition the configuration of the terminal is performed by **RFID_Command_Session_SetTerminalType** command.

TRFID_Terminal object is used as the command parameter. It registers the declared terminal type (one of **eRFID_TerminalType** constants) and the set of the required access rights to the electronic document capabilities (combination of **eRFID_TerminalAuthorizationRequirement**) [24, part 3, §C.4]. Moreover, it can be done either manually by initializing in a corresponding way TermType, AuthReq and AuthReq2 fields, or automatically, by specifying the full name of the respective terminal certificate file, containing all necessary information, in `TermCert_FileName` field (or placing its contents in `TermCert_Data` field). In the second case, **`TRFID_Terminal`** structure will be automatically initialized on the basis of the certificate contents in the process of command execution.

> *Note.* **`RFID_Command_Session_SetTerminalType`** command requires no mandatory presentation of a pointer to the open session object as a parameter and may only be used to initialize **`TRFID_Terminal`** object by information contained in the given terminal certificate.

Attempting to perform operation unauthorized for the current terminal type, in the further work with the document the user software will receive an error code as a return code from the SDK function, corresponding to the detected situation.

The terminal configuration established this way is registered in `Session_terminal` field of **`TRFID_Session`** open session object.

5.8.5. Authentication Procedure Type Definition

The second *mandatory* operation after opening of electronic document working session is the operation of **authentication procedure type definition** (see section 4.11). To perform this operation there is **`RFID_Command_Session_SetProcedureType`** command. One of **`eRFID_AuthenticationProcedureType`** values serves as the command parameter.

The type of ongoing authentication procedure affects some aspects of different SDK functions (for example, the order of PACE performance for `aptGeneral`).

However, it used more for self-organization of user software to ensure strict compliance of its operational logics with the standardized electronic document work procedures [1, 24].

The type of ongoing authentication procedure determined this way is registered in `Session_procedure` field of **`TRFID_Session`** open session object.

5.8.6. Protected Data Access Key Definition

The third *mandatory* operation after opening of electronic document working session is the operation of **protected data access key definition** (see section 4.4).

It is used to set type and value of the key used for organization of secure communication channel between the reader and the RFID-chip when implementing the SM mechanism.

To perform this operation there is **RFID_Command_Session_SetAccessKey** command. A pointer to **TRFID_AccessKey** object serves as the command parameter, specific fields of which contains the given:

- accessType – type of the basic secure data access mechanism (acptBAC or acpt-PACE);
- keyType – access key type (one of **eRFID_Password_Type** values);
- AccessKey – key value.

When working with *eSign* application this command can be used for *eSign-PIN* initialization as well. In this case, the contents of accessType are ignored, and eSignPIN_Index field must contain the identifier of the used *eSign-PIN* (in SDK version 3.1 only the value 1 is supported).

When working with *ePassport* application CheckFullKeyMatching field can contain a logical sign of the need for additional comparison of the full contents of AccessKey with the contents of DG1 data group (MRZ).

The protected data access key defined this way is registered in Session_key field of **TRFID_Session** open session object.

5.8.7. Authentication Procedures Performance

To perform all types of procedures of authentication and secure data access there is **RFID_Command_Session_AccessControlProc** command.

A pointer to **TRFID_AccessControl_Params** object, which identifies the type of the performed procedure (ac_Type field), and its parameters (ac_Params fields), is used as the command parameter.

Depending on the type of procedure when executing the command ac_Params contents are interpreted in different ways:

- for BAC/BAP and AA the contents are ignored;
- for PACE field must contain a pointer to **TPACE_SetupParams** object;
- for CA – a pointer to **TCA_SetupParams** object;
- for TA – a pointer to **TTA_SetupParams** object;
- for RI – a pointer to **TRI_SetupParams** object.

All information accompanying the performed procedure will be registered in the corresponding **TRFID_AccessControlInfo** object – element of pAccessControls list of the active session object.

If successful, the return code from SDK function of the procedure performance will be **RFID_Error_NoError** value.

The beginning of the operation is marked with sending **RFID_Notification_ACProcedure_Start** notification message to the user application, its ending – with **RFID_Notification_ACProcedure_Finish**. The low order WORD of these notification codes contains the type of performed procedure (a value from **eRFID_AccessControl_ProcedureType**), message parameter – the result of procedure (a value from **eRFID_ErrorCodes**). The same result code is registered in Status field of the corresponding **TRFID_AccessControlInfo** object.

The index of the active variant of the procedure is registered in ActiveOptionIdx field of **TRFID_AccessControlInfo** object.

5.8.8. Organization of Secure Data Access Channel

One of the main requirements for electronic documents is to implement a mechanism for data exchange between reader and chip using secure communication channel.

BAC/BAP and PACE are the basic procedures for organization of such communications channel.

RFID_LAYER6_SECURITY_MANAGER error code, which means the requirement of organization of secure communication channel, will be returned when trying to read data protected in this way, within the active session.

For PACE, ac_Params field of **RFID_Command_Session_AccessControlProc** command parameters must contain a pointer to **TPACE_SetupParams** object, which determines the chosen variant of the procedure in the list of available variants (i.e., in the list TRFID_AccessControlInfo.pOptions of the corresponding pAccessControls element of the current session object):

- index of the variant is specified in nOptionIdx field;
- skipCHAT – a sign that it is necessary to transfer CHAT data to the chip when initializing the procedure (see section 4.7).

Note. The use of CHAT is *mandatory* only if TA shall be used after PACE. In other cases (for example, when performing the procedure of temporary PIN resuming using CAN password) CHAT usage is not recommended (i.e., skipCHAT must contain false).

If successful, the return code from SDK function of the procedure performance will be **RFID_Error_NoError** value.

Any other error code will point to one or another problem that might appear, including those due to incorrect value of the protected data access key. In the latter case, the user application may, for instance, provide an opportunity to use a different key value or try to use the key of another type.

ATTENTION! In accordance with the requirements [24] SM-channel in the case of *PACE*, must be organized *prior to holding any communication with the chip* apart from `EF.CardAccess` reading (i.e., immediately after session opening and the performance of three mandatory operations to its configuration, see sections 5.8.3–5.8.6). In case of BAC– *after executing the command of* `ePassport` *application selection*.

Detection of the requirement for opening a secure communication channel is accompanied by **RFID_Notification_SM_Required** notification message to the user application.

The user application is notified of the result of SM channel opening (`true` or `false`) by **RFID_Notification_SM_Established** notification message.

If necessary, procedure of SM organizing will be performed after the first unsuccessful attempt to read the protected data automatically, using the current assigned access key value. In this case, the determination of the appropriate variant of PACE procedure is as follows:

- based on the contents of `DefaultPACEOptionIdx` field of **TRFID_AccessKey** structure used when specifying data access key (see section 5.8.6),
- when processing **RFID_Notification_SM_Required** notification by setting the index of appropriate variant of procedure by **RFID_Command_Set_DefaultPACEOption** command.

5.8.9. Application Selection

To access a particular function of the electronic document or to a file in its memory, it is required to select the corresponding application first.

There is **RFID_Command_Session_SelectApplication** command for this.

A pointer to **TRFID_ApplicationID** object, containing an identifier of the selected application, is used as the command parameter. A zero value is used to select the root *Master File* of the document.

In case of a successful operation a corresponding element will be stored in `pApplications` list of the session object. `ActiveApplicationIdx` field of the session object will contain the index of the currently selected application in the list (or `-1` for *Master File*).

For any outcome of the operation the user application will be informed by **RFID_Notification_PCSC_ApplicationSelected** notification, containing an application code in the low order `WORD` (one of **eRFID_Application_Type** values).

Note. Identifiers of all supported by SDK standard applications are given in the module `RFID_Common.cpp/.h`.

5.8.10. File Reading

The command **RFID_Command_Session_ReadFile** used to read the contents of a file belonging to the currently selected application of the electronic document.

A pointer to **TRFID_FileID** object containing file description is used as the command parameter:

- pID field – file identifier;
- nLength – file identifier length;
- id_type – file type (one of **eRFID_FileID_Type** values);
- SM_protected – an indicator that access to the file should be organized through a secure SM-channel;
- FixedLength – fixed file length if it is known in advance or required a specific number of bytes to be read (0 – file length is automatically determined by the length of the title tag of its contents in ASN.1 format).

> *Note.* **TRFID_FileID** descriptions for all supported by SDK standard files are given in the module RFID_Common.cpp/.h.

In the process of file data reading the following actions are performed:

- formation of a binary array with the contents of read data;
- if necessary – automatic attempt to start BAC for the organization of a secure communication channel;
- a logical analysis of the data to form a list of detected text or graphic document fields or the formation (updating) of sets of the corresponding service session objects (document security objects, variants of performing authentication procedures etc.);
- registration of critical and non-critical remarks of logical data analysis and performance of actions along with sending the respective notification messages to the user application;
- formation of the final result of the command.

TRFID_DataFile object, containing the results of these actions, is stored in pFiles list of the respective pApplications element of the session object. To store file objects from the root *Master File* pRootFiles list of the session object is used.

The beginning of the reading of any file and its end is marked by sending **RFID_Notification_PCSC_ReadingDatagroup** message. A code of read file (contained in the low order WORD) from **eRFID_DataFile_Type** enumeration is introduced in the code of this notification.

Sending **RFID_Notification_Progress** message marks a progress of command execution. The numeric value passed with the message, determines the amount of data read as a percentage of the total amount of file data.

To interrupt the current operation there is **RFID_Command_CancelReading** command.

Depending on the type of file being read pParsedData field of **TRFID_DataFile** object may contain a pointer to different data objects, which describe in detail the logical structure of the file contents.

Field value **nType**	**Pointer type** **pParsedData**	**Field value** **nType**	**Pointer type** **pParsedData**
dftPassport_COM dftDL_COM	TRF_EFCOM *	dftPassport_DG16	TRF_EF_DG16 *
dftPassport_DG1	TRF_EF_DG1 *	dftApp_Directory	TRFID_Items_List *
dftPassport_DG2 dftPassport_DG3 dftPassport_DG4 dftDL_DG6 dftDL_DG7 dftDL_DG8	TRF_EF_DG234 *	dftID_DG1 dftID_DG2 dftID_DG3 dftID_DG4 dftID_DG5 dftID_DG6 dftID_DG7 dftID_DG8 dftID_DG10 dftID_DG11	TRF_EID_TEXT_ARRAY *
dftPassport_DG5 dftPassport_DG6 dftPassport_DG7 dftDL_DG5	TRF_EF_DG567 *	dftID_DG9 dftID_DG17	TRF_EID_GENERAL_PLACE *
dftPassport_DG8 dftPassport_DG9 dftPassport_DG10	TRF_EF_DG8910 *	dftID_DG12 dftID_DG21	TRF_EID_OPTIONAL_DATA *
dftPassport_DG11	TRF_EF_DG11 *	dftID_DG13 dftID_DG14 dftID_DG15 dftID_DG16 dftID_DG18	TRF_EF_DG_BINARY_ARRAY *
dftPassport_DG12	TRF_EF_DG12 *	dftID_DG19 dftID_DG20	TRF_EID_TEXT *
dftPassport_DG13	TRF_EF_DG_BINARY_ARRAY *	dftDL_DG1	TRF_EDL_DG1 *

For other types of files pParsedData is used solely for the internal SDK operation.

nStatus fields of the respective elementary fields of logical data representation (**TRF_FT_BYTE**, **TRF_FT_WORD**, **TRF_FT_NUMBER**, **TRF_FT_BYTES**, **TRF_FT_STRING**) may contain codes of detected inconsistencies of the contents to the requirements of one or another specifications – the value from **eLDS_ParsingNotificationCodes** or **errLDS_Ok**.

A set of informational data groups present in *ePassport* application can be defined in two ways:

- by the contents of EF.COM. For this it is required to read its contents and analyze the acquired list of identifiers of the present data groups TRF_EFCOM.bDataGroup;
- by the contents of EF.SOD. After reading EF.SOD it is possible to check the presence of one or another data group hash value in its structure by special **RFID_Command_Session_PA_IsFileCheckAvailable.** command. A pointer to the file identifier **TRFID_FileID** serves as its parameter.

As a result of such analysis for *ePassport* and *eDL* applications an additional element of dftApp_Directory type included into pFiles list, which pParsedData contains a list of identifiers of information data groups that are present in the application, automatically compiled on the basis of the combination of read EF.COM and EF.SOD contents.

When working with the *eID* application there is no single method for determining the presence of a file other than a direct attempt to read it. This is due to the fact that the analogue of EF.COM (object storing a list of all present informational data groups) is not provided for the *eID* application by the standards [24] and [25], and the presence of data group hash tables in the document security objects is not mandatory (as presence of hash values for all present data groups in this table).

In case of a situation where the requested file is missing, the user application will be informed by **RFID_Notification_PCSC_FileNotFound** notification message with a file code in the low order WORD.

In case of reaching the end of file prior to acquisition of all the requested amount of data the user application will be informed by **RFID_Notification_PCSC_EndOfFile** notification message with a file code in the low order WORD.

In case if the terminal has no access rights to the requested file, the user application will be informed by **RFID_Notification_PCSC_FileAccessDenied** notification message with a file code in the low order WORD.

5.8.11. Data Reading According to MIFARE® Classic Protocol

RFID_Command_Session_ReadMifare command used to read data from the chip that supports communications protocol by the standard ISO/IEC 14443-3 (MIFARE® Classic Protocol).

The result is the formation of two elements:

- *read data*;
- *arrays of flags of reading correctness for each chip memory sectors*.

Corresponding **TRFID_DataFile** objects are stored in pRootFiles list of the current session object . The data type is specified in nType field of TRFID_DataFile structure:

- `dftMIFARE_Data` – for data;
- `dftMIFARE_Validity` – for flags of reading correctness.

Example: with chip memory amount 1 Kb, `dftMIFARE_Data` array will contain data of 1024 bytes, and `dftMIFARE_Validity` array – 16 bytes with the value `0`, if the corresponding sector was read with error, or `1`, if the data were read successfully.

In the process of command execution, the user application will receive **`RFID_Notification_ReadProtocol3`** message, informing about the beginning and the end of data reading operation.

5.8.12. Passive Authentication: Document Security Object Verification

While reading of service files as part of electronic document communication session, detected document security objects are being registered in `pSecurityObjects` list of active session. The copies of **`TRFID_SecurityObject`** are the elements of this list.

Each **`TRFID_SecurityObject`** contains a list of its corresponding digital signature objects `pSignerInfos`, which in the most cases contains the only **`TRFID_SignerInfo_Ex`** element.

To check the document security objects as a part of passive authentication (see section 4.8.1) there is **`RFID_Command_Session_PA_CheckSO`** command.

A pointer to **`TPA_Params`** structure, containing a description of the inspected SO, is used as the command parameter:

- field `SO_Index` – index of the inspected SO in `pSecurityObjects` list of the active session object;
- `SI_Index` – index of the inspected digital signature in `pSignerInfos` list of SO object.

During command execution the following actions are performed:
- search for DS-certificate for SO digital signature verification;
- search for CSCA-certificate for DS- certificate digital signature verification;
- structure analysis of the located certificates and their digital signatures verification;
- SO digital signature verification;
- registration of critical and non-critical remarks, arising during the actions, along with sending the respective notification messages to the user application.

Certificate search for SO digital signature verification, the order of their use and CSCA-certificate trust level accepted here correspond to the current parameters of PA (see section 5.5.2).

The need to use one of the variants of public key search (for `Issuer` and `serialNumber` or `subjectKeyIdentifier`, see section 4.8.1) for the specific digital signature object is defined by the contents of the respective fields in **TRFID_SignerInfo_Ex** structure.

For the user application there is a possibility to provide a set of certificates for SO check directly to the input of the command. For this there are the corresponding fields of **TPA_Params** structure. In this case, the user application is responsible for the full validation of the certificates presented this way. On the side of the SDK it is accepted on default that the presented certificates are deliberately authentic and have maximum trust level.

The beginning and the end of the procedure of formation of the corresponding certificate chain (see section 4.8.1) is accompanied by sending **RFID_Notification_PA_CertificateChain** notification, a type of the current processing element of the chain is indicated by **RFID_Notification_PA_CertificateChainItem** notification.

The result of the check is stored in `PA_Status` field of the respective **TRFID_SignerInfo_Ex** object, information about the used certificate chain – in `pCertificateChain` field.

The user application will be informed about the operation outcome by **RFID_Notification_PA_SecurityObjectCheck** notification, containing in the low order `WORD` a file type, on the basis of which inspected SO has been built (one of **eRFID_DataFile_Type** values). The message parameter will contain the result of the procedure (corresponding value from **eRFID_ErrorCodes**).

5.8.13. Passive Authentication: Data Informational Groups Integrity Verification

The second stage of the procedure of passive authentication is to verify the integrity of informational data groups of the electronic document (see section 4.8.1). It is performed by comparison of the computed hash values of actually read data with the values contained in the document security objects and thus protected by using digital signature mechanism.

To perform this operation there is **RFID_Command_Session_PA_CheckFile** command.

As the command parameter a pointer **TRFID_DataFile *** to the structure with description of the verifiable file, which is an element of `pFiles` file list of the respective application of the current session, is used.

The result of verification is stored in `PA_Status` field of the respective **TRFID_DataFile** object.

The user application will be informed about the operation outcome by **RFID_Notification_PA_FileCheck** notification, containing a file type in the low order WORD (one of **eRFID_DataFile_Type** values). The message parameter will contain the result of the procedure (corresponding value from **eRFID_ErrorCodes**).

> **ATTENTION!** In accordance with the requirements [2] and [3] the informational data groups integrity check for the *ePassport* application is *mandatory*. When working with the *eID* application this procedure is *not mandatory*, because the list of individual hash values may be absent in the presented document security objects [24] or may be incomplete [25].

RFID_Command_Session_PA_IsFileCheckAvailable command may be used as the auxiliary command, by which it is possible to check the presence of a hash value of a particular data group in the structure of detected document security objects. A pointer to the file identifier **TRFID_FileID *** serves as its parameter. RFID_Error_NoError return code indicates a presence of the respective value (it means, of potential presence of the data group itself in the memory of electronic document), **RFID_Error_NotAvailable** code – its absence.

5.8.14. Chip Authentication Procedure

To perform the chip authentication procedure (see section 4.9.2) ac_Params field in the parameters of **RFID_Command_Session_AccessControlProc** command must contain a pointer to **TCA_SetupParams** object, which defines the chosen variant of performing the procedure from the list of available variants (i.e., in TRFID_AccessControlInfo.pOptions list of the respective pAccessControls element of the current session object):

- Index of variant is given in the field nOptionIdx,
- TA_preliminary_step – an indicator to perform the TA preliminary stage as a part of the EAC (version 2), which requires creation of ephemeral CA cryptographic keys pair (see section 4.9.3).

Thus, when implementing the EAC (version 2), it is required to execute the command of CA performance *twice*. The first time – as a part of TA preliminary step (TA_preliminary_step = true), right before TA procedure itself, the second time – to perform all the necessary operations of data exchange with the RFID-chip as a part of CA.

> **ATTENTION!** In both calls, the command must be given the same index of variant of procedure performance.

In case of successful procedure performance the return code from the SDK function will be **RFID_Error_NoError** value.

5.8.15. Terminal Authentication Procedure

TA procedure consists of three consequent steps:

- formation of certificate chain in accordance with the index of the selected variant of procedure performance (actually – in accordance with the selected identifier of the public CVCA-key) and its verification by the RFID-chip;
- formation of special data token (*challenge*) and its digital signature generation using the terminal private key;
- *challenge's* digital signature verification by the RFID-chip.

To perform terminal authentication procedure (see section 4.9.3) `ac_Params` field in the parameters of **`RFID_Command_Session_AccessControlProc`** command must contain a pointer to **`TTA_SetupParams`** object, which defines the chosen variant of performing the procedure from the list of available variants (i.e., in `TRFID_AccessControlInfo.pOptions` list of the respective `pAccessControls` element of the current session object) and the parameters of procedure performance:

- index of variant is given in the field `nOptionIdx`;
- `ProcessType` – the order of procedure performance (one of **`eRFID_TerminalAuthenticationType`** values);
- `PACE_StaticBinding` – a sign of PACE static binding, determines the method of the data token composition for the digital signing [24, part 1, §3.5];
- `TA_StepData` – configuration of another TA step when working in step-by-step mode without using the callback-function;
- `VerificationData` – contents of auxiliary data provided for the following verification (see section 4.10.2).

Several variants of TA procedure performance are possible, which are defined by the corresponding constant in `ProcessType` field:

- *default* mode (`tatDefault`), when execution of all procedure steps takes place automatically, using the available resources of the local PKD (see sections 4.9.3, 5.5.3);
- *Online-authentication* mode (`tatOnline`), when call of user callback-function precedes each step with request of the respective parameters;
- *interruptible step-by-step* operation mode (`tatStepByStep`), when return from the SDK function occurs after execution of each step (call of callback-function is not performed). In this variant execution of the next TA step means only a repeated execution of the command **`RFID_Command_Session_AccessControlProc`** with updated input data, which are necessary for the next step of procedure.

In both variants of step-by-step TA performance the data for another step of procedure are transferred by **`TTerminalAuthenticationStepData`** structure.

With *Online-authentication* a pointer to this structure is transferred in the parameter of **`RFID_Notification_TA_Step`** notification message. The user application must initialize the respective fields of structure by the given pointer and only after this perform return from the callback-function, providing thus execution of the next step.

In case of *interruptible step-by-step mode* the input data of each procedure step are transferred by contents of `TA_StepData` field of the input command parameters .

The identifier of the required public CVCA-key becomes known to the user application from the contents of `KeyAlgorithm` field of the selected TA procedure performance variant (see section 5.8.3). It will be also stored in `TTerminalAuthenticationStepData.CAR` field with the first arrival of **RFID_Notification_TA_Step** notification.

When organizing step-by-step work the user application must independently perform search of the corresponding certificates and terminal private key and transfer these data to the SDK by the individual fields of **TTerminalAuthenticationStepData** structure provided for it.

For the first step of TA procedure repeat iterations (**RFID_Notification_TA_Step** notifications or returns from the SDK function) will occur as long as the user application does not pass null pointers for the input of data, which will be a signal to move to the next step procedure. This, for example, enable the use of the required number of CVCA-Link-certificates.

After successful completion of certificate chain verification by the RFID-chip the *challenge* data will be composed. `Challenge` field in **TTerminalAuthenticationStepData** will describe its contents.

If the user application is not required to obtain a signed data itself but its already computed hash (for [36] requirements), this option can be selected with **RFID_Command_Set_OnlineTAToSignDataType** command. In this case, the hash value will be placed in the field `Challenge` too.

Having received these data during the second step of procedure, the user application can independently generate digital signature and store its value in the **TTerminalAuthenticationStepData** (`Signature` field).

Another variant for this is to specify the data of terminal private key during the first step. In this case *challenge's* digital signature may be formed by the SDK. For this the user application must keep the contents of `Signature` empty.

The third TA step is performed automatically, without additional requests to the user software.

> *Note.* Responsibility for allocating and freeing memory for storing the input data of TA procedure in **TA_StepData** structure lies on the user's application.

In case of successful procedure performance the return code from the SDK function will be **RFID_Error_NoError** value.

ATTENTION! In case of failure of TA procedure without PACE static binding retrying with the option of static binding will be carried out automatically.

All detected critical and non-critical remarks during the process of logical data analysis (including the data specified by the user) and other actions performed during the procedure, will be registered in the respective **TRFID_AccessControl_Option** along with a notification of the user application by notification messages.

5.8.16. Active Authentication Procedure

When performing active authentication procedure (see section 4.8.2), the contents of the ac_Params field in the parameters of **RFID_Command_Session_AccessControlProc** command are ignored. Here it means that the variant of the AA performance can be only one and it is determined by the contents of EF.DG15 informational data group of *ePassport* application.

In case of successful procedure performance the return code from the SDK function will be **RFID_Error_NoError** value.

5.8.17. Restricted Identification Procedure

To perform the procedure of restricted identification (see section 4.10.1) ac_Params field in the parameters of **RFID_Command_Session_AccessControlProc** command must contain a pointer to the object **TRI_SetupParams**, which defines the chosen variant of procedure performance from the list of available variants (i.e., in TRFID_AccessControlInfo.pOptions list of the respective pAccessControls element of the current session object) and the parameters of procedure performance.

Index of variant is given in the field nOptionIdx.

The contents of RI public keys to be sent to the chip are specified in SectorKey1 and SectorKey2 fields, or names of the respective files with public keys data are given in SectorKey1_FileName and SectorKey2_FileName fields.

Key contents representation is allowed in a form of ASN.1-object SubjectPublicKeyInfo [6, § 4.1] or in a form of the respective public key data TLV-object [24, § D.3].

In case of successful procedure performance the return code from the SDK function will be **RFID_Error_NoError** value.

The result of command execution (terminal sector identifiers) is stored in SpecificData1 and SpecificData2 fields of **TRFID_AccessControlInfo** object, corresponding to the RI, in pAccessControls list of the current session object.

After reception of each of two possible terminal sector identifiers from the RFID-chip call of callback-function of the user application takes place with **RFID_Notification_RI_SectorID** notification code, containing identifier type in the low order WORD (one of **eRFID_SectorKeyType** values). A pointer to the respective **TRF_FT_BYTES** object with identifier contents is transferred as the function parameter.

Processing this call the user application is able to perform search of sector identifier in the local revocation database (see section 4.10.1) and specify the result of this search (**RFID_Error_Failed**, **RFID_Error_NotAvailable** or **RFID_Error_NoError**) in nType field of **TRF_FT_BYTES** object. Herewith completeness and logical integrity of the results of work with electronic document is preserved within the context of the current session. By default nType field contains RFID_Error_NotPerformed value.

5.8.18. Auxiliary Data Verification

To perform auxiliary data verification within the context of TA procedure (see section 4.10.2) there is **RFID_Command_Session_Verify** command. One of **eRFID_AuxiliaryDataType** values serves as its parameter, which defines the type of data to be verified.

Auxiliary data verification of each specific type becomes available only if the respective data were defined in the input parameters of TA procedure (in VerificationData field of **TTA_SetupParams** structure – see section 5.8.15).

The contents of verified data are stored in VerifiedData field of the current session object, which is **TTerminalVerificationData** object, the result of command execution (**RFID_Error_Failed**, **RFID_Error_NoError** or by default **RFID_Error_NotPerformed**) – in nType field of the respective subfield of **TRF_FT_BYTES** in VerifiedData.

The user application is notified about the results of verification by **RFID_Notification_AuxiliaryDataValidation** notification message, containing the result code and the type of verified data (in the low order WORD) in the notification parameter.

5.8.19. Data Informational Group Contents Update (eID application)

For the terminal having relevant rights according to the results of effective authorization (see section 4.7), there is a possibility to update the contents of DG17–DG21 informational data groups the *eID* application.

RFID_Command_Session_WriteFile command serves for updating a file with new contents. A pointer to **TRFID_FileUpdateData** structure serves as its parameter, defining the type of file to be updated (FileID field) and its new contents (Data field).

In case of successful procedure performance the return code from the SDK function will be **RFID_Error_NoError** value.

5.8.20. Password Management

For the terminal having relevant rights according to the results of effective authorization (see section 4.7), there is a possibility to execute a number of management functions for protected data access keys (see section 4.5.1) or for *eSign-PIN* management (see section 4.5.2).

The following are included in the set of respective commands:

- **RFID_Command_Session_Password_ChangePIN** – change of PIN. A pointer (`char *`) to the character string with a new password value is transferred as the command parameter;
- **RFID_Command_Session_Password_ChangeCAN** – change of CAN. A pointer (`char *`) to the character string with a new password value is transferred as the command parameter;
- **RFID_Command_Session_Password_UnblockPIN** – PIN unblocking;
- **RFID_Command_Session_Password_ActivatePIN** – PIN activation;
- **RFID_Command_Session_Password_DeactivatePIN** – PIN deactivation;
- **RFID_Command_Session_eSign_CreatePIN** – *eSign-PIN* generation;
- **RFID_Command_Session_eSign_ChangePIN** – *eSign-PIN* change;
- **RFID_Command_Session_eSign_UnblockPIN** – *eSign-PIN* unblocking;
- **RFID_Command_Session_eSign_TerminatePIN** – *eSign-PIN* terminating.

In all commands of *eSign-PIN* management a pointer to **TRFID_eSignPINParameters** object serves as an input parameter, containing key identifier in `PIN_Id` field (for SDK version 3.1 only value `1` is supported), and in `PIN_new` field – the new password d value (for commands of *eSign-PIN* change and creation).

> *Note.* The working *eSign-PIN* value is assigned by **RFID_Command_Session_SetAccessKey** command (see section 5.8.6) and is automatically stored in **Session_eSignPIN** field of the current session object.

In case of successful procedure performance the return code from the SDK function will be **RFID_Error_NoError** value.

5.8.21. eSign Application Management and Usage

For the terminal having relevant rights according to the results of effective authorization (see section 4.7), there is a possibility to execute a number of functions on management of `eSign` application and use of its function of data digital signature.

RFID_Command_Session_eSign_GenerateKeyPair command serves for creation of `eSign` cryptographic keys pair, **RFID_Command_Session_eSign_TerminateKeyPair** command – for its termination (deactivation of `eSign` application). The parameter of both commands is a pointer to **TRFID_eSignKeyParameters** structure, containing identifier of a key pair in the key_Id field (in SDK version 3.1 only the value 1 is supported).

The contents of the created public key are stored as **TRFID_DataFile** object in pFiles (list of *eSign* application files) of the respective element of pApplications list of the current session object. Here, the value dft_eSign_PK is given as the file type (TRFID_DataFile.nType).

The operation of *eSign-PIN* verification must precede the procedure of data digital signature generation. For its performance there is **RFID_Command_Session_eSign_VerifyPIN** command. A pointer to **TRFID_eSignPINParameters** object serves as the input parameter, containing key identifier in PIN_Id field (for SDK version 3.0 only the value 1 is supported). The current value of verified *eSign-PIN* is assigned by **RFID_Command_Session_SetAccessKey** command (see section 5.8.6).

In case of successful performance of *eSign-PIN* verification (the return code from the SDK function will be **RFID_Error_NoError** value) it becomes possible to execute the command of the data digital signature generation – **RFID_Command_Session_eSign_SignData**. **TCustomRawData** structure is the command parameter, containing data to be signed.

The contents of the generated digital signature are stored as **TRFID_DataFile** object in pFiles (list of *eSign* application files) of the respective element of pApplications list of the current session object. Here, dft_eSign_SignedData value is given as the file type (TRFID_DataFile.nType). The digital signature data are stored in TRFID_DataFile.FileData, and the data to be signed – in TRFID_DataFile.FileID.

5.8.22. Saving and Loading of Work Session Data

SDK has a possibility of representing a complex data structure of session work (**TRFID_Session** object) as the integral memory block. It gives a possibility, for example, to save results of session work in a file and then reload them from there, receiving a possibility of full data analysis at any time, outside the context of a real communication session with electronic document.

To present **TRFID_Session** object as the integral memory block there is **RFID_Command_Session_SaveData** command.

A pointer to **TRFID_Session** working session object serves as the input parameter of the command (parameter result of **_RFID_ExecuteCommand()** function).

TCustomRawData * – a pointer to the container of binary data is used as the output parameter of the command (parameter params of **_RFID_ExecuteCommand()** function).

TCustomRawData.buffer must contain a pointer to a memory area dedicated for receipt of data of session object, TCustomRawData.length – the length of dedicated memory fragment.

If the command input receives **TCustomRawData** with zero buffer value or the size of presented memory block is insufficient for acceptance of all data, the required size of memory block will be specified in length field when returning from the function. In this case the user application may perform allocation of the given memory amount and repeatedly execute the command for acquisition of correct result.

To create **TRFID_Session** object on the basis of the existing integral block of data there is **RFID_Command_Session_LoadData** command.

TCustomRawData * – a pointer to the source of binary data is used as the input parameter of a command (parameter params of **_RFID_ExecuteCommand()** function).

An address of the pointer to the created session object (**TRFID_Session ****) serves as the output parameter of the command (parameter result of **_RFID_ExecuteCommand()** function).

TRFID_Session object created this way will fully correspond to the results of the original communication session with electronic document. VirtualMode field will contain the value TRUE, indicating a «virtuality» of the session under presentation. Here, no SDK session commands will be available for it.

> *Note.* Memory allocation for storing of integral data block always takes place on the side of the calling application.

5.9. Scenario Operation Mode

5.9.1. Working Scenario

5.9.1.1. Scenario Structure

SDK scenario is an XML-based structure which defines the basic parameters of ongoing session of work with the electronic document:

- terminal configuration (see section 5.8.4),
- authentication procedure type (see section 5.8.5),
- secure data access key (see section 5.8.6),
- set of the reading data groups (see sections 5.8.9, 5.8.10),
- the need of performing the procedures of restricted identification (see section 5.8.17) and auxiliary data verification (see section 5.8.18)

etc.

The structure of the scenario is of the form:

```
<arbitrary_root_node_name>
   <parameter_node_0>value</parameter_node_0>
   ...
   <component_parameter_node_i>
     <nested_parameter_node_0>
     ...
     <nested_parameter_node_j>
   </component_parameter_node_i>
   ...
   <parameter_node_N>значение</parameter_node_N>
</arbitrary_root_node_name>
```

Scenario parameters can be integral, containing a single value, and composite, consisting of several nested parameters. They are divided into several groups that define a particular aspect of the functioning of the SDK during the session of work with the electronic document (see section 5.9).

> *Note.* In SDK version 3.4 scenario mode provides support for *a session to* ***read*** *data only*. Support for working with `eSign` application, as well as password management operations and ability to update `eID` application data group content is absent in this mode.

5.9.1.2. Terminal Type Definition

Parameters used to define the terminal configuration when initializing **`TRFID_Terminal`** structure (see sections 5.8.4, 6.3.83):

`<TerminalManualConfig>`
logical sign of a manual terminal characteristics definition – `true` (manual) / `false` (automatic)

`<TerminalCertificate>`
file name of the CV-certificate, on the basis of which the characteristics of the terminal will be determined (for automatic mode)

`<TerminalType>`
terminal type (numeric value of the corresponding constant)

`<UniversalAccessRights>`
logical sign of the use of the universal access rights to the capabilities of electronic document («all inclusive» mode). If `false`, set of access rights will be composed on the basis of the following options:

- rights for reading data groups (*ePassport*, *eID* applications) formed on the basis of actually requested in the current reading session (see below)
- rights to use the functionality of the electronic document for AT terminal (see section 4.7) – based on the following parameters (all values of `true`/`false`)

```
<Authorized_Install_QCert>
<Authorized_Install_Cert>
<Authorized_PIN_Managment>
<Authorized_CAN_Allowed>
<Authorized_PrivilegedTerminal>
<Authorized_RestrictedIdentification>
<Authorized_Verify_CommunityID>
<Authorized_Verify_Age>
<Authorized_Write_DG17>
<Authorized_Write_DG18>
<Authorized_Write_DG19>
<Authorized_Write_DG20>
<Authorized_Write_DG21>
```

- rights to use the functionality of the electronic document for ST terminal (see section 4.7) – based on tthe following parameters (all values of `true`/`false`)

```
<Authorized_ST_Signature>
<Authorized_ST_QSignature>
```

5.9.1.3. Authentication Procedure Type Definition

Parameters used to define the authentication procedure type (see section 5.8.5):

`<AuthProcType>`
authentication procedure type, conducting which is expected in the presence of all the objective conditions

5.9.1.4. Base Secure Data Access Channel Mechanism Definition

Parameters used to define the priority mechanism for organizing the secure SM-channel:

`<BaseSMProcedure>`
type of priority mechanism for organizing SM-channel

5.9.1.5. Secure Data Access Key Definition

Parameters used to define the secure data access key (see section 5.8.6):

`<PACEPasswordType>`
data access type for PACE procedure (numeric value of the corresponding constant)

`<MRZ>`
full document MRZ text in accordance with requirements set in section 5.7.6

`<Password>`
text string of the data access key, other than MRZ (CAN, PIN, PUK)

`<FullMRZMatching>`
logical sign of the need for additional comparison of the full contents of the given MRZ string with the contents of DG1

`<AlwaysAskForMRZ>`
logical sign of the need to request a custom application to provide the data access key before its actual use in the organization of SM-channel

5.9.1.6. Terminal Authentication Procedure Parameters

Parameters used during the procedure of terminal authentication (see section 5.8.15):

`<OnlineTA>`
logical sign of the Online-authentication performance

`<PACE_StaticBinding>`
logical sign of forced use of static binding with PACE

5.9.1.7. Verifiable Auxiliary Data Definition

Parameters used during the procedure of auxiliary data verification (see section 5.8.18):

`<AuxVerification_CommunityID>`
logical sign of the need to verify Community ID

`<AuxVerification_DateOfBirth>`
logical sign of the need to verify the age of the document holder

`<AuxVerification_DateOfExpiry>`
logical sign of the need to verify the validity period of the document

`<AuxVerification_CommunityID_Data>`
string containing verifiable `Community ID` data. The string format - hex image of the contents, 2 characters per byte, with no spaces (for example – «`40414243`» to ASCII-character string «`ABCD`»)

`<AuxVerification_DateOfBirth_Data>`
string containing the verifiable date in format YYYYMMDD

5.9.1.8. Passive Authentication Procedure Parameters

Parameters used during the procedure of passive authentication (see sections 5.8.12, 5.8.13):

`<PassiveAuth>`
logical sign of passive authentication performance

5.9.1.9. Active Authentication Procedure Parameters

Parameters used during the procedure of active authentication (see section 5.8.16):

`<SkipAA>`
logical sign of the cancellation of active authentication procedure after the successful performance of chip authentication (CA) procedure

5.9.1.10. Restricted Identification Procedure Parameters

Parameters used during the procedure of restricted identification (see section 5.8.17):

`<Perform_RestrictedIdentification>`
logical sign of restricted identification procedure performance

`<SectorPKCertificate1>`
first RI public key filename

`<SectorPKCertificate2>`
second RI public key filename

5.9.1.11. Definition of the Set of Informational Data Group to Read

Parameters used to define the set of informational data groups (see section 5.8.10):

```
<ePassport>
      <DG1>true/false</DG1>
      ...
      <DG16>true/false</DG16>
</ePassport>
```
for `ePassport` application

```
<eID>
      <DG1>true/false</DG1>
      ...
      <DG21>true/false</DG21>
</eID>
```

for `eID` application

5.9.1.12. Parameters for Data Reading According to ISO/IEC 14443-3

Parameters used during the data reading according to ISO/IEC 14443-3 standard (see section 5.8.11):

```
<Read_3>
```

logical sign of the need to read data

5.9.2. Scenario Composition

The scenario of the SDK generated by user application in accordance with the specified configuration.

To aid in the formation of scenario XML-structure **`RFID_SDK_UI.dll`** library serves, exporting special **`_RFID_UI_Helper_ManageSetup()`** helper function. One of its parameter is a sign of the need to display a window that allows visually specify the contents of the script in a dialogue with the user. As another parameter `VARIANT *setup_xml_var` pointer appears, which points to a text string of scenario XML-structure. If provided `setup_xml_var` is empty, the scenario will be created with default values for all parameters. If it already contains the correct scenario XML-structure – it contents will appear in the calling dialog exactly. When finished using the setup dialog box scenario XML-structure with the specified parameters will be placed in `setup_xml_var` container to return to the calling application.

> *Note.* All operations of memory initialization and reallocation with the VARIANT held by the SDK and should be performed by the user application using the appropriate functions of Windows API.

The library **`RFID_SDK_UI.dll`** developed for the dynamic connection using Windows API **`LoadLibrary()`** function. Pointers to exported functions can be obtained using Windows API **`GetProcAddress()`** function.

After loading the library into memory it is required to make a call to the initialization function **`_RFID_UI_Helper_Initialize()`**.

At the end of the work with the library it is required to call **`_RFID_UI_Helper_Free()`** function and unload the library from memory using Windows API **`FreeLibrary()`** function.

5.9.3. Scenario Execution

To run the scenario there is **`RFID_Command_Scenario_Process`** command. As a parameter it takes `char *` pointer to a text string of XML-scenario (UTF8 coding supported).

When running the script there is a certain sequence of actions additionally concerted with the parameters specified in the scenario XML-structure:

- opening the session, searching and reading `EF.CardAccess` file to determine whether PACE procedure supported, the variants for its conduct (see section 5.8.3)
- setting the terminal type (see section 5.8.4)
- setting the authentication procedure type (see section 5.8.5). Since the possibility of one or another procedure type depends on objective conditions (presence of certain files, support for specific versions of secure data access procedures by the electronic document, etc.), if the specified authentication procedure cannot be performed, *the rollback mechanism to the next available variant* is used.
- setting the secure data access key (see section 5.8.6)
- conducting PACE procedure if it is supported by the electronic document and when chosen as a priority mechanism for SM-channel organization (see section 5.8.8)
- conducting version 2 TA procedure (see section 5.8.15) and the preliminary step of version 2 CA procedure (see section 5.8.14). Setting auxiliary data for verification (see section 5.8.18)
- in the case of general authentication procedure – searching and reading `EF.CardSecurity` и `EF.ChipSecurity` files, an update on their basis a list of available CA procedure variants, verification of the digital signatures of the corresponding document security objects as a part of PA procedure (see section 5.8.12)
- in the case of general authentication procedure – conducting version 2 CA procedure (see section 5.8.14)
- conducting restricted identification procedure (see section 5.8.17)
- for AT terminal – auxiliary data verification (see section 5.8.18)
- selecting *`ePassport`* application, searching and reading `EF.SOD` file, an update on its basis a list of available informational data groups (see section 5.8.10), verification of the digital signature of the corresponding document security object as a part of PA procedure (see section 5.8.12)
- in the case when BAC selected as a base SM mechanism, the procedure of opening a secure data access channel conducted upon the first attempt of secure data reading (`EF.COM`) (see section 5.8.8)
- searching and reading `EF.COM` и `EF.DG1` files (regardless of the user's choice). Updating the list of informational data groups available for reading. Errors of reading data files are critical, and, if they occur, further work is interrupted

- searching and reading `EF.DG14` file (regardless of the user's choice), an update on its basis a list of available variants of version 1 CA procedure
- conducting version 1 CA procedure (see section 5.8.14)
- checking the integrity of previously read informational data groups (DG1, DG14) as a part of PA procedure (see section 5.8.13)
- searching and reading `EF.DG15` file, checking its data integrity as a part of PA procedure, determination of the capability and the variant of AA procedure, conducting AA procedure (see section 5.8.16)
- searching and reading of informational data groups files defined by user (if actually available) that are not protected by EAC mechanism, their data integrity check as a part of PA procedure (see section 5.8.13)
- if version 1 TA needed – searching and reading `EF.CVCA` file to determine the public key identifier. Conducting version 1 TA procedure (see section 5.8.15)
- searching and reading of EAC-sensitive informational data groups files defined by user (if actually available) that are not protected by EAC mechanism, their data integrity check as a part of PA procedure
- selecting *`eID`* application, searching and reading of informational data groups files defined by user, their data integrity check as a part of PA procedure when corresponding hash-values are present in `EF.CardAccess` (see sections 5.8.10, 5.8.13)
- selecting *`eDL`* application, searching and reading `EF.SOD` file, an update on its basis a list of available informational data groups (see section 5.8.10), verification of the digital signature of the corresponding document security object as a part of PA procedure (see section 5.8.12)
- in the case when BAP selected as a base SM mechanism, the procedure of opening a secure data access channel conducted upon the first attempt of secure data reading (`EF.COM`) (see section 5.8.8)
- searching and reading `EF.COM` и `EF.DG1` files (regardless of the user's choice). Updating the list of informational data groups available for reading. Errors of reading data files are critical, and, if they occur, further work is interrupted
- checking the integrity of previously read informational data groups (DG1) as a part of PA procedure (see section 5.8.13)
- searching and reading of informational data groups files defined by user (if actually available) that are not protected by EAP mechanism, their data integrity check as a part of PA procedure (see section 5.8.13)
- data reading according to MIFARE® Classic Protocol.

As an output parameter of **`RFID_Command_Scenario_Process`** command also acts `char **` pointer, which will point to the XML-representation of **`TRFID_Session`** structure filled with the results of the data reading session (see section 5.8.2).

Furthermore, as in the session mode, after the command completion the contents of the corresponding **TRFID_Session** object will be stored in the list of results as the element with dftSession type for RFID_ResultType_RFID_BinaryData data representation type (see section 5.6.2).

5.9.4. Scenario Requests

5.9.4.1. Structure and Mechanics of the Request

During the scenario execution, when the number of points that require or allow user (user application) interaction reached, the call of notification callback-function is performed with **RFID_Notification_Scenario** notification code. A pointer char ** containing a pointer to the XML-string defining a concrete scenario step acts as a parameter (memory management is on the side of SDK library). Data required by SDK from the user application on a specific step are to be placed into the same scenario step XML-structure and passed back through the contents of the same char ** (memory management is on the side of the user application).

These steps of the scenario are:

- the need to change access key to the protected data and/or its value that occurs when an attempt to organize SM-channel failed or when the user application demands such request explicitly, before SM-channel establishing,
- the need to choose the variant of one or another authentication or secure data access procedure (PACE, CA, TA, RI),
- the possibility or need to use user data in the implementation of the authentication or secure data access procedure (TA, RI, PA).

In case when the user application cannot provide the required data or for some reason returns an empty scenario step XML-string, further SDK actions are determined based on their default order: either available data or settings used or reading operation interrupted (in the case of secure data access key requests).

SDK scenario request to the user application is an XML-structure of the following format:

```
<SDK_Scenario_Step>
   <SDK_Request Type="request_type_identifier">
     <request_parameter_0>
     ...
     <request_parameter_N>
   </SDK_Request>
</SDK_Scenario_Step>
```

Returned in response to a request data should be placed in the XML-structure as follows:

```
<SDK_Scenario_Step>
   <return_data_0>
     ...
   <return_data_N>
</SDK_Scenario_Step>
```

5.9.4.2. Selection of the Authentication Procedure / Secure Data Access Variant

Request type identifier:	"**AC_Option_Selection**"
Request parameters:	
`<AC_Type>`	abbreviation of the constant identifying the type of procedure (from **eRFID_AccessControl_ProcedureType** enumeration, for example "`acptPACE`", "`acptTA`" etc.)
`<CurrentACOptionIdx>`	the current index of the selected variant of procedure conducting
`<RFID_AccessControl_Option>`	**TRFID_AccessControl_Option** structure (XML-representation, see section 6.3.70) for each available variant of procedure conducting
Return data:	
`<ActiveOptionIdx>`	the index of the selected variant of procedure conducting

5.9.4.3. Request for the Secure Data Access Key

Request type identifier:	«**AccessKey**»
Request parameters:	
`<TerminalType>`	numeric value of the current terminal type (from **eRFID_TerminalType** enumeration)
`<AuthReq>` `<AuthReq2>`	current combination of the access rights flags to the functionality of the electronic document (see section 6.3.83)
`<PACEPasswordType>`	numeric value of the current data access key type (from **eRFID_Password_Type** enumeration)
`<PwdManagementAction>`	numeric value of the current operation with the data access key (from **eRFID_PasswordManagementAction** enumeration)
`<BlockedPassword>`	numeric value of the key type (from **eRFID_Password_Type** enumeration), blocked

	for the current operation – for each element included in the set of blocked types

Return data:

`<PACEPasswordType>`	numeric value of the key type (from **eRFID_Password_Type** enumeration) selected for use
`<FullMRZMatching>`	logical sign of the need for additional comparison of the full contents of the given MRZ string with the contents of `DG1`
`<MRZ>`	document MRZ full text, in accordance with the requirements set out in section 5.7.6
`<Password>`	text string of the data access key other than MRZ (CAN, PIN, PUK, SAI)

5.9.4.4. Request for the Action on the Secure Data Access Key

Request type identifier:	" **KeyManagement**"
Request parameters:	
`<PACEPasswordType>`	numeric value of the current key type (from **eRFID_Password_Type** enumeration)

`<PwdManagementStatus>`	numeric value of the key status – one of the values from **eRFID_ErrorCodes** enumeration:
	`RFID_LAYER6_SECURITY_MANAGER`
	`RFID_LAYER6_PWD_SUSPENDED`
	`RFID_LAYER6_PWD_SUSPENDED_2`
	`RFID_LAYER6_PWD_BLOCKED`
	`RFID_LAYER6_PWD_BLOCKED_2`
	`RFID_LAYER6_PWD_DEACTIVATED`
	`RFID_LAYER6_PWD_DEACTIVATED_2`

Return data:

`<PwdPostAction>`	numeric value of the selected action with the key (from **eRFID_PasswordPostDialogAction** enumeration)

5.9.4.5. Request for the Certificate Chain for Passive Authentication Procedure

Request type identifier:	"**PA_Resources**"
Request parameters:	
`<Issuer>`	**TRFID_DistinguishedName** structure (XML-representation, see section 6.3.75), containing an issuer identifier (see sections 4.8.1, 5.8.12)
`<SerialNumber>`	serial number the required certificate – CDATA node, containing Base64-encoded binary data

`<SubjectKeyIdentifier>`	subject identifier of the required certificate – CDATA node, containing Base64-encoded binary data

Return data:

```
<PA_Certificate>
   <Data>
     <![CDATA[]]>
   </Data>
</PA_Certificate>
```

DER-encoded certificate data, for each certificate included in the chain – CDATA node, containing Base64-encoded binary data.

5.9.4.6. Request for the Certificate Chain for Terminal Authentication Procedure

Request type identifier:	"**`TA_Resources`**"
Request parameters:	
`<CAR>`	required public CVCA-key identifier (see section 5.8.15)

Return data:

```
<TA_Certificate>
   <Data>
     <![CDATA[]]>
   </Data>
   <PrivateKey>
     <![CDATA[]]>
   </PrivateKey>
</TA_Certificate>
```

certificate and the corresponding private key data for each certificate included in the chain – CDATA nodes, containing Base64-encoded binary data.

5.9.4.7. Request for the Digital Signature of the Challenge for Terminal Authentication Procedure

Request type identifier:	"**`TA_Signature`**"
Request parameters:	
`<Challenge>`	the contents of the control data fragment (see section 5.8.15) – CDATA node, containing Base64-encoded binary data
`<HashValue>`	the hash value of the control data fragment (see section 5.8.15) – CDATA node, containing Base64-encoded binary data

Return data:

```
<TA_Signature>
   <Data>
     <![CDATA[]]>
   </Data>
```

`</TA_Signature>`	digital signature of the challenge data – CDATA node, containing Base64-encoded binary data.

5.9.4.8. Request for the Status of the Terminal Sector Identifier for Restricted Identification Procedure

Request type identifier:	"**`RI_Status`**"
Request parameters:	
`<Sector_ID>`	terminal sector identifier (see section 5.8.17) – CDATA node, containing Base64-encoded binary data
Return data:	

`<Sector_ID_Status>`	numeric value of the status of the terminal sector identifier – одно from **`eRFID_ErrorCodes`** enumeration:
	`RFID_Error_NoError`
	`RFID_Error_Failed`

6. SDK SOFTWARE TOOLS

6.1. EXPORTED FUNCTIONS

All exported functions are declared with the specificator **`extern «C»`**.

6.1.1. _RFID_Initialize()

Type:	**`typedef DWORD (*_RFID_Initialize)(DWORD)`**
Symbolic name:	**`_RFID_Initialize`**
Assignment:	initialization of the main control library
Parameters:	combination of **`eRFID_ControlRF`** values

During call of this function search, loading and initialization of functional libraries necessary for work is performed as well as the formation of a list of RFID-chip readers available in the system.

Return code – one of **`eRFID_ErrorCodes`** values:

- `RFID_Error_NoError` – operation completed successfully;
- `RFID_Error_AlreadyDone` – function was called already; the library is ready for further work.

Any return value other than `RFID_Error_NoError` or `RFID_Error_AlreadyDone` means that further work is impossible because of critical errors. In this case, it is necessary to call **`_RFID_Free()`** deinitialization function, unload the `DLL` from the memory and finish the application, or by taking the necessary measures to restore the capability, to repeat the call to the initialization function.

6.1.2. _RFID_Free()

Type:	**`typedef DWORD (*_RFID_Free)()`**
Symbolic name:	**`_RFID_Free`**
Assignment:	deinitialization of the main control library

Calling this function release of all resources of the main control library takes place. To resume work it is necessary to call the **`_RFID_Initialize()`** again.

Return code – one of **`eRFID_ErrorCodes`** values:

- `RFID_Error_NoError` – operation completed successfully;
- `RFID_Error_AlreadyDone` – function was already called.

ATTENTION! **`_RFID_Free()`** and **`_RFID_Initialize()`** must be called from the same thread of the user application.

6.1.3. _RFID_SetCallbackFunc()

Type: **`typedef void (*_RFID_SetCallbackFunc)(RFID_NotifyFunc func)`**

Symbolic name: **`_RFID_SetCallbackFunc`**

Assignment: setting of callback-function for receiving messages on the status of command execution, on changes of the internal status of the library or the device.

This function as the parameter accepts a pointer to the function of the user application, which will be called during the process of command execution, when special situations arise or there are changes in the device state.

6.1.4. _RFID_ExecuteCommand()

Type: **`typedef DWORD (*_RFID_ExecuteCommand)(int command, void *params, void *result)`**

Symbolic name: **`_RFID_ExecuteCommand`**

Assignment: request to command execution

Parameters:

- `command` – command code (one of **`eRFID_Commands`** constants)
- `params` – input parameters of command
- `result` – output parameters of command (results container)

Return code – one of **`eRFID_ErrorCodes`** values:

- `RFID_Error_NoError` – operation completed successfully;
- `RFID_Error_NotInitialized` – control library is not initialized (there was no **`_RFID_Initialize()`** call);
- `RFID_Error_InvalidParameter` – incorrect input parameter of function is passed;
- `RFID_Error_PCSC_CardIsBusy` – execution of previous command is not completed;
- `RFID_Error_PCSC_ReaderNotAvailable` – no active RFID-chip reader;
- `RFID_Error_NoChipDetected` – RFID-chip is absent in the scope of the reader;
- `RFID_Error_UnknownCommand` – unknown command etc.

Sending of all the commands, defining and reading of device performance parameters is done using **`_RFID_ExecuteCommand()`** function passing the command code, by transfer of command parameters and – for reading commands – a pointer to a container for receiving the results.

The types of input parameters and result container are defined on assumption of the context of each specific command.

6.1.5. _RFID_CheckResult()

Type: **`typedef HANDLE (*_RFID_CheckResult)(DWORD type, DWORD output, void *param)`**

Symbolic name: **`_RFID_CheckResult`**

Assignment: acquisition of the container with results of data reading from the RFID-chip memory by the type of their representation; data conversion into XML format

Parameters:

`type` – type of the requested result (one of the types of data representation **`eRFID_ResultType`**)

`output` – additional results representation (**`eOutputFormat`**)

`param` – parameters for additional result representation

Return value:

- if > 0, it is a descriptor of the structure containing data of the requested type (**`TResultContainer *`**, casted to the type `HANDLE`);
- if < 0 – one of **`eRFID_ResultStatus`** values.

This function must be called after the command of RFID-chip data reading.

The user application must free memory occupied by these data after receiving the result through **`_RFID_CheckResult()`**, by calling **`_RFID_FreeResult()`**.

There are two ways of application of this function:

1) the conversion of result data containers into XML format;
2) the reception of a descriptor of result data containers for further access to its separate fields with the help of **`_RFID_CheckResultFromList()`** function (for **`RFDP_FullyParsed`** result representing type).

To form the XML-representation it is necessary to pass one of `ofClipboard_XML`, `ofFile_XML` or `ofXML` values as `type` parameter, A pointer to a text buffer containing the resulting XML-document will be placed in `XML_buffer` field of the return `TResultContainer` structure, and its length – in `XML_length` field.

When `type=ofClipboard_XML`, text of the resulting XML-document will be placed in Windows clipboard in `CF_TEXT` format (see the documentation on programming in Windows environment) and will be available after return from **`_RFID_CheckResult()`**. The clipboard will be associated with a window, a handle of which (`HWND`) has been passed to `param`.

When `type=ofFile_XML`, text of the resulting XML-document will be saved in a text file. A pointer to a text string (`char *`) containing the full file name in UTF8 format must be passed to `param`.

6.1.6. _RFID_CheckResultFromList()

Type: **typedef int (*_RFID_CheckResultFromList) (HANDLE container, DWORD output, void *param)**

Symbolic name: **_RFID_CheckResultFromList**

Assignment: access to separate fields of read data result structure (for **RFDP_FullyParsed** type of result representation)

Parameters:

- `container` – descriptor of the result received by **_RFID_CheckResult()**
- `output` – additional result representation (**eOutputFormat**)
- `param` – parameters for additional result representation

Return result:

- if >= `0`, it is a digital code of the field to be transferred (one of **eVisualFieldType**, **eRFID_VisualFieldType** or **eGraphicFieldType** values);
- if < `0` – one of **eRFID_ResultStatus** values:
  1) `RFID_ResultStatus_EndOfList` – the end of the fields list in the result structure is reached (field value, transferred with the previous function call, was the last in the list);
  2) `RFID_ResultStatus_InvalidParameter` – incorrect function parameter;
  3) `RFID_ResultStatus_Error` – error of formation of additional result representation (when saving the file or when placing to the clipboard).

When assigning the `output=offInfo` for the field of any type no additional actions with data will be performed.

When working with **text fields** (`RFID_ResultType_RFID_TextData`) the following actions are performed:

- `output = offClipboard`

In this case, the contents of the current text field are stored in Windows clipboard in `CF_TEXT` format and will be available after return from the function (see the documentation on programming in Windows environment).

When working with **graphic fields** (`RFID_ResultType_RFID_ImageData`) the logics of the performed actions will be as follows:

- conditional test
  `output = offClipboard`

In this case, the contents of the current graphic field are entered in Windows clipboard in `CF_DIB` format and will be available after return from the function (see the documentation on programming in Windows environment). Later return from the function occurs.

- conditional test

  `(output & offFile) = true`

or

  `output = offFileBuffer`

For the image from the current field a graphic file contents image of the assigned type is formed. The type of graphic format is defined by the file name extension.

With `(output & offFile)=true` a full name of the required graphic format is passed through `param` parameter, which should contain a pointer to a text string with the name (`char *`) in UTF8 format. In this case, the data will be written in the file specified.

With `output=offFileBuffer`, `param` parameter should contain a pointer **`TResultContainer *`**, serving as a container of the result being formed. The file name of the required graphic format (or just the appropriate extension) should be in the field `XML_buffer` of this object.

- conditional test

  `(output & offXML) = true`

XML-representation of graphic file image is formed.

- conditional test

  `(output & offClipboard) = true`

The result (XML-representation of graphic file image) is stored in Windows clipboard in `CF_TEXT` format. Later return from the function occurs.

- conditional test

  `output = offFileBuffer`

A pointer to the data array of the formed file image is stored in `buffer` field, its length – in `buf_length` field. Later return from the function occurs.

After reaching this point the saving of generated data in a graphic file under the specified name and return from the function is taking place.

The logic of performed actions when working with the **original images of graphic fields** (`RFID_ResultType_RFID_OriginalGraphics`) is as follows:

- conditional test

  `output = offFile`

Type of the original graphic format is defined by the contents of `GraphicsType` of **`TOriginalRFIDGraphics`** object of the current field (one of **`eRFID_OriginalGraphicsType`** values).

The contents of the field are written to a file, a full name of which has been passed through `param` parameter, containing a pointer to a text string with the name (`char *`) in UTF8 format. This file's extension is replaced by the default one, corresponding to a particular original format. Later return from the function occurs.

- conditional test

  `output = offFileBuffer`

In this case, `param` parameter must contain a pointer **`TResultContainer *`**, acting as a result container.

A pointer to the data array of original file image is stored in `buffer` field, its length – in `buf_length` field. Later return from the function occurs.

> *Note.* A complete list of supported graphic file formats can be obtained by using **`RFID_Command_Get_AvailableGraphicFormats`** command.

In case of work with `gf_Portrait` graphic field there is a possibility of its automatic modification during the process of execution of **`_RFID_CheckResultFromList()`** – cast to the specified image height. To specify the desirable height in pixels there is **`RFID_Command_Set_CheckResultHeight`** command.

6.1.7. _RFID_LibraryVersion()

Type:	**`typedef DWORD (*_RFID_LibraryVersion)()`**
Symbolic name:	**`RFID_LibraryVersion`**
Assignment:	returns the version of the main SDK control library: `HIWORD()` – major version, `LOWORD()` – minor version (for the version 3.2 – 3 and 2 respectively)

6.1.8. _RFID_UI_Helper_Initialize()

Type:	**`typedef DWORD (*_RFID_UI_Helper_Initialize)(DWORD)`**
Symbolic name:	**`_RFID_UI_Helper_Initialize`**
Parameters:	reserved
Assignment:	initializing the helper library for scenarios management

When calling this function initialization of resources required for the helper library occurs.

Return code – one of **`eRFID_ErrorCodes`** values:

- `RFID_Error_NoError` – operation completed successfully;
- `RFID_Error_AlreadyDone` – function was called already, library is ready for further work.

6.1.9. _RFID_UI_Helper_Free()

Type: **`typedef DWORD (*_RFID_UI_Helper_Free)()`**
Symbolic name: **`_RFID_UI_Helper_Free`**
Assignment: deinitializing the helper library for scenarios management

When calling this function deinitialization of used by the helper library resources occurs. To restart it is necessary to make **`_RFID_UI_Helper_Initialize()`** call again.

Return code – one of **`eRFID_ErrorCodes`** values:

- `RFID_Error_NoError` – operation completed successfully;
- `RFID_Error_AlreadyDone` – function was called already.

6.1.10. _RFID_UI_Helper_ManageSetup()

Type: **`typedef DWORD (*_RFID_UI_Helper_ManageSetup)(VARIANT *setup_xml_var, HANDLE hWnd)`**
Symbolic name: **`_RFID_UI_Helper_ManageSetup`**
Assignment: scenario XML-structure formation and management of its parameters
Parameters:

`setup_xml_var` – the text content of the scenario XML-structure
`hWnd` – logical sign of the need to display a window that allows visually specify the contents of the script in a dialogue with the user

In case `setup_xml_var` value is empty (`bstrVal` field of `VARIANT` structure contains 0), scenario XML-structure will be formed containing the default parameters.

Further, if given a non-zero `hWnd` parameter, a dialog box will be displayed to control scenario parameters visually.

Thus formed scenario XML-structure will be placed in `setup_xml_var` container to return to the calling application.

in case of displaying dialog box, the value of the return code will depend on the order of closing the window. When it closed by clicking "***Ok***" button the return code will be `RFID_Error_NoError`, "***Cancel***" button – `RFID_Error_Failed`.

In operation without dialog window with correct input parameters the return code will always be equal to `RFID_Error_NoError`.

6.2. CALLBACK-FUNCTION

This is the function of the user application, which is called by the main control library for the notification on the status of execution of commands or on changes of library or device internal statuses. Its use is set by using **`_RFID_SetCallbackFunc()`** function (see section 5.1).

The type of callback-function is declared in **`RFID.h`**:

```
typedef void (_stdcall *RFID_NotifyFunc)(int code, void *value);
```

Accordingly, in the user application it must be declared like:

```
void _stdcall MyNotifyFunc(int code, void *value);
```

and set as follows:

```
RFID_SetCallbackFunc(MyNotifyFunc);
```

Parameters:

`code`	–	notification code (**`eRFID_NotificationCodes`**)
`value`	–	value (in the context of the code)

6.3. STRUCTURES

6.3.1. TResultContainerList

TResultContainerList structure is used to store and transfer to the user application complete list of various types of representation of the read data when executing **RFID_Command_ReadProtocol3** and **RFID_Command_ReadProtocol4** commands (see section 5.6.2).

```
struct TResultContainerList
{
  DWORD            Count;
  TResultContainer *List;
};
```

Declaration: `PasspR.h`

Fields:

`Count`	–	number of `List` array elements
`List`	–	array of containers for data of different type of representation.

6.3.2. TResultContainer

TResultContainer structure is used to store results of read data from the RFID-chip for one type of data representation and is a generating structure for **TResultContainerList**.

```
struct TResultContainer
{
  DWORD   result_type;
  DWORD   light;
  DWORD   buf_length;
  void   *buffer;
  DWORD   XML_length;
  BYTE   *XML_buffer;
  DWORD   list_idx;
  DWORD   page_idx;
};
```

Declaration: `PasspR.h`

Fields:

`result_type`	–	identifier defining the type of pointer stored in `buffer`. A value – one of **eRFID_ResultType** identifiers
`light`	–	not used
`buf_length`	–	size of the data structure referenced to by `buffer`
`buffer`	–	pointer to a structure with the results of data reading (a specific type of data is determined by `result_type` field value)
`XML_length`	–	size of `XML_buffer` array, in bytes

`XML_buffer` – text array containing representation of structure with results of data reading in XML format
`list_idx` – for internal use
`page_idx` – for internal use

Value of **`result_type`**	Pointer type **`buffer`**
`RFID_ResultType_RFID_RawData`	`TDocBinaryInfo *`
`RFID_ResultType_RFID_TextData`	`TDocVisualExtendedInfo *`
`RFID_ResultType_RFID_ImageData`	`TDocGraphicsInfo *`
`RFID_ResultType_RFID_BinaryData`	`TDocBinaryInfo *`
`RFID_ResultType_RFID_OriginalGraphics`	`TOriginalRFIDGraphics *`

`XML_length` and `XML_buffer` fields are initialized only when calling **`_RFID_CheckResult()`** function passing one of `ofClipboard_XML`, `ofFile_XML` or `ofXML` requested result types (see section 5.6.3).

6.3.3. TDocBinaryInfo

`TDocBinaryInfo` structure is used to store the data reading results from the RFID-chip in a form of a list of the logically separated data groups. It is used for **`RFID_ResultType_RFID_RawData`** and **`RFID_ResultType_RFID_BinaryData`** result representing types.

```
struct TDocBinaryInfo
{
   DWORD        nFields;
   TBinaryData *pArrayFields;
};
```

Declaration: `PasspR.h`
Fields:
`nFields` – number of `pArrayFields` array elements
`pArrayFields` – array of structures for different logically separated data groups

6.3.4. TBinaryData

`TBinaryData` structure is a basic structure for **`TDocBinaryInfo`** list.

```
struct TBinaryData
{
   int  FieldType;
   char FieldName[256];
   int  Buf_Length;
   BYTE *Buffer;
};
```

Declaration: `PasspR.h`

Fields:

`FieldType` – type of data group that is stored in this container (one of **`eRFID_DataFile_Type`** identifiers)

`FieldName` – data group symbolic name

`Buf_Length` – size of the data structure referenced to by `Buffer`

`Buffer` – pointer to the data group structure

Depending on the context `Buffer` may contain a pointer to different structures:

- a simple byte array (of `Buf_Length` length), containing the exact copy of a specified data group, stored in the memory of RFID-chip without additional formatting with all the service information (separation tags, etc.) for **`RFID_ResultType_RFID_RawData`** type;
- a structure with the description of data group, the type of which is defined by `FieldType` value for **`RFID_ResultType_RFID_BinaryData`** type.

6.3.5. TDocVisualExtendedInfo

`TDocVisualExtendedInfo` structure is used to store the results of data reading from the RFID-chip in a form of a list of logically separated text data (text fields). It is used for **`RFID_ResultType_RFID_TextData`** type of data representation.

```
struct TDocVisualExtendedInfo
{
   int                         nFields;
   TDocVisualExtendedField *pArrayFields;
};
```

Declaration: `PasspR.h`

Fields:

`nFields` – number of `pArrayFields` array elements

`pArrayFields` – array of the structures containing logically separated text data

6.3.6. TDocVisualExtendedField

`TDocVisualExtendedField` structure is the basic container structure for **`TDocVisualExtendedInfo`** the and stores information about a single text data field (see sections 5.7.2, 5.10, 6.1.6).

```
struct TDocVisualExtendedField
{
   long                FieldType;
   long                RFID_OriginDG;
   long                RFID_OriginDGTag;
   long                RFID_OriginTagEntry;
   long                RFID_OriginEntryView;
```

```
  char              FieldName[256];
  int               StringsCount;
  TStringResultSDK *StringsResult;
  int               Buf_Length;
  char             *Buf_Text;
  char             *FieldMask;
  int               Validity;
  int               InComparison;
  DWORD             Reserved2;
  DWORD             Reserved3;
};
```

Declaration: `PasspR.h`

Fields:

`wFieldType`	–	logical type of text field (one of **`eVisualFieldType`** or **`eRFID_VisualFieldType`** values)
`wLCID`	–	not used
`RFID_OriginDG`		source file of the text field (**`eRFID_DataFile_Type`**)
`RFID_OriginDGTag`	–	not used (always contains `0`)
`RFID_OriginTagEntry`	–	not used (always contains `0`)
`RFID_OriginEntryView`	–	not used (always contains `0`)
`FieldName`	–	symbolic name of the text field
`StringsCount`	–	not used
`StringsResult`	–	not used
`Buf_Length`	–	length of the text string in `Buf_Text`
`Buf_Text`	–	string with text data of the field in `UTF8` format
`FieldMask`	–	string of format mask of text data of the field
`Validity`	–	not used
`InComparison`	–	not used
`Reserved2`	–	not used
`Reserved3`	–	not used

XML-representation of the structure:

```
<RFID_Text_Field>
   <FieldType Text=""> – numeric FieldType value
   <OriginDG Text="">  – numeric RFID_OriginDG value
   <OriginDGTag>       – numeric RFID_OriginDGTag value
   <OriginTagEntry>    – numeric RFID_OriginTagEntry value
   <OriginEntryView>   – numeric RFID_OriginEntryView value
   <Buf_Text>          – string Buf_Text value
   <FieldMask>         – string FieldMask value
</RFID_Text_Field>
```

`Text` attributes of `FieldType` and `OriginDG` nodes contain text abbreviations of the corresponding values.

6.3.7. TDocGraphicsInfo

TDocGraphicsInfo structure is used to store the results of data reading from the RFID-chip in a form of a list of logically separated graphic data (images, graphic fields). It is used for **RFID_ResultType_RFID_ImageData** results representation type.

```
struct TDocGraphicsInfo
{
  int              nFields;
  TDocGraphicField *pArrayFields;
};
```

Declaration: PasspR.h

Fields:

nFields – number of pArrayFields array elements

pArrayFields – array of images

6.3.8. TDocGraphicField

TDocGraphicField structure is a basic container structure for **TDocGraphicsInfo** list and contains information about a single graphic field.

```
struct TDocGraphicField
{
  int                FieldType;
  long               RFID_OriginDG;
  long               RFID_OriginDGTag;
  long               RFID_OriginTagEntry;
  long               RFID_OriginEntryView;
  char               FieldName[256];
  TRawImageContainer image;
};
```

Declaration: PasspR.h

Field:

FieldType – logical type of graphic field (one of **eGraphicFieldType** values)

RFID_OriginDG – source file of the image (**eRFID_DataFile_Type**)

RFID_OriginDGTag – index of source record of the image with biometric information in the informational data group

RFID_OriginTagEntry – index of the template in the record with biometric data

RFID_OriginEntryView – index of the variant of the biometric data template

FieldName – symbolic name of the graphic field

image – image data

XML-representation of the structure:

```
<RFID_Graphic_Field>
   <FieldType Text="">   – numeric FieldType value
   <OriginDG Text="">    – numeric RFID_OriginDG value
   <OriginDGTag>         – numeric RFID_OriginDGTag value
   <OriginTagEntry>      – numeric RFID_OriginTagEntry value
   <OriginEntryView>     – numeric RFID_OriginEntryView value
   <File_Image>
      <Length>           – size of the graphic image file
      <Format>           – file extension of the graphic format (".bmp"), used for
                           the image encoding
      <Data>
         <![CDATA[]]>    – Base64-encoded byte array of the graphic image file
      </Data>
   </File_Image>
</RFID_Graphic_Field>
```

`Text` attibutes of `FieldType` and `OriginDG` nodes contain text abbreviations of the corresponding values.

6.3.9. TRawImageContainer

`TRawImageContainer` structure is used to store and transfer of graphic images in uncompressed Windows DIB format to the user application.

```
struct TRawImageContainer
{
  BITMAPINFO  *bmi;
  BYTE        *bits;
};
```

Declaration: `PasspR.h`

Fields:

`bmi` – Windows DIB header with 256-color palette (if the format of image provides palette)

`bits` – image array of pixels (`DWORD` aligned)

The amount of memory allocated for bmi, equals to

`sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD)*256`.

The amount of memory allocated for bits, equals to `bmi.bmiHeader.biSizeImage`.

6.3.10. TOriginalRFIDGraphicsInfo

`TOriginalRFIDGraphicsInfo` structure is used to store the results of data reading in a form of a list of objects of the original binary representation of the graphics in memory of the RFID-chip. It is used for **RFID_ResultType_RFID_OriginalGraphics** type of results representation.

```
struct TOriginalRFIDGraphicsInfo
{
  DWORD                  nFields;
  TOriginalRFIDGraphics *pArrayFields;
};
```

Declaration: `RFID.h`

Fields:

`nFields`	–	number of `pArrayFields` array elements
`pArrayFields`	–	array of images

6.3.11. TOriginalRFIDGraphics

TOriginalRFIDGraphics structure is a basic container structure for **TOriginalRFIDGraphicsInfo** list and contains information about a single object of the original binary representation of the graphics in memory of the RFID-chip.

```
struct TOriginalRFIDGraphics
{
  int  FieldType;
  int  GraphicsType;
  int  RFID_OriginDG;
  int  RFID_OriginDGTag;
  int  RFID_OriginTagEntry;
  int  RFID_OriginEntryView;
  int  Buf_Length;
  BYTE *Buffer;
};
```

Declaration: `RFID.h`

Fields:

`FieldType`	–	logical type of graphic field (one of **eGraphicFieldType** values)
`GraphicsType`	–	image encoding type (**eRFID_OriginalGraphicsType**);
`RFID_OriginDG`	–	source file of the image (**eRFID_DataFile_Type**)
`RFID_OriginDGTag`	–	index of source file of the image with biometric information in the informational data group
`RFID_OriginTagEntry`	–	index of the template in the biometric data record
`RFID_OriginEntryView`	–	index of the variant of the biometric data sample
`Buf_Length`	–	length of `Buffer` array
`Buffer`	–	object data binary array

XML-representation of the structure:

```
  <RFID_OriginalGraphic_Field>
     <FieldType Text=""> – numeric FieldType value
     <GraphicsType Text=""> – numeric GraphicsType value
     <OriginDG Text="">  – numeric RFID_OriginDG value
     <OriginDGTag>       – numeric RFID_OriginDGTag value
     <OriginTagEntry>    – numeric RFID_OriginTagEntry value
     <OriginEntryView>   – numeric RFID_OriginEntryView value
     <File_Image>
        <Length>         – numeric Buf_Length value
        <Data>
           <![CDATA[]]>  – Base64-encoded Buffer contents
        </Data>
     </File_Image>
</RFID_OriginalGraphic_Field>
```

`Text` attibutes of `FieldType`, `OriginDG` and `GraphicsType` nodes contain text abbreviations of the corresponding values.

6.3.12. TRFID_CardPropertiesExt

`TRFID_CardPropertiesExt` structure is used to store extended information about the characteristics of the RFID-chip located in the scope of the reader (see sections 5.7.1, 5.8.3).

```
struct TRFID_CardPropertiesExt
{
  TRFCardProp Properties;
  DWORD       cbAtr;
  BYTE        pAtr[36];
};
```

Declaration: `RFID.h`

Fields:

`Properties` – information about characteristics of the RFID-chip

`cbAtr` – length of `pAtr` array

`pAtr` – `ATR` string of the RFID-chip

XML-representation of the structure (see also section 6.3.13):

```
  <CardProperties>
    <RFID_Type>        – text abbreviation of RFID_Type value
    <Baudrate1>        – numeric Baudrate1 value in hexadecimal format (e.g.
                         "0x0000000F")
    <Baudrate2>        – numeric Baudrate2 value in hexadecimal format
    <Support_4>        – boolean Support_4 value
    <ChipType_A>       – text abbreviation of ChipType_A value
    <Support_Mifare>   – boolean Support_Mifare value
```

`<MifareMemory>`	–	numeric `MifareMemory` value
`<UID>`	–	`UID` contents in text format. Each byte is represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"F9 4F 41 60"`)
`<ATQ_A>`	–	numeric `ATQ_A` value in hexadecimal format (e.g. `"0x0000"`)
`<SAK>`	–	numeric `SAK` value in hexadecimal format (e.g. `"0x00"`)
`<ATQ_B>`	–	`ATQ_B` contents in text format. Each byte is represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"50 F9 4F 41 60 00 00 00 00 77 81 81"`)
`<BitRateS>`	–	numeric `BitRateS` value in hexadecimal format (e.g. `"0x04"`)
`<BitRateR>`	–	numeric `BitRateR` value in hexadecimal format (e.g. `"0x04"`)
`<ATR>`	–	`pAtr` contents in text format. Each byte is represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"3B 88 81 11 FC 00 00 00 00 77 81 81 00 93"`)

`</CardProperties>`

6.3.13. TRFIDCardProp

`TRFCardProp` structure is used to store information about characteristic of the RFID-chip located in the scope of the reader (see sections 5.7.1, 5.8.3).

```
struct TRFCardProp
{
  DWORD  RFID_Type;
  WORD   Baudrate1;
  WORD   Baudrate2;
  BOOL   Support_4;
  DWORD  ChipType_A;
  BOOL   Support_Mifare;
  DWORD  MifareMemory;
  BYTE   SizeUID;
  BYTE   UID[10];
  WORD   ATQ_A;
  BYTE   SAK;
  BYTE   ATQ_B[13];
  BYTE   BitRateS;
  BYTE   BitRateR;
};
```

Declaration: `RFID.h`

Fields:

`RFID_Type` – type of the RFID-chip by the physical parameters of the connection between chip and reader antennas (one of **`eRFID_Type`** constants)

`Baudrate1` – combination of **`eRFID_BaudRate`** flags, defining the data transmitting rates supported by the RFID-chip

`Baudrate2` – combination of **`eRFID_BaudRate`** flags, defining the data receiving rates supported by the RFID-chip

`Support_4` – sign of support for ISO/IEC 14443-4 data exchange protocol – `true` or `false`

`ChipType_A` – type of the chip from the MIFARE® family, supporting ISO/IEC 14443-3 protocol (MIFARE® Classic Protocol) (for chips of the «`A`» type) – one of **`eRFID_A_Chip`** constants

`Support_Mifare` – sign of support for ISO/IEC 14443-3 data exchange protocol (MIFARE® Classic Protocol) – `true` or `false`

`MifareMemory` – amount of operational memory MIFARE® of the chip, kilobytes

`SizeUID` – length of `UID` field

`UID` – unique chip identifier

`ATQ_A` – reply of the «`A`» type chip to «REQA» command of ISO/IEC 14443-3 protocol (*Answer To Request*, *Type A – ATQA*) – for the internal use by the main control library

`SAK` – reply of the «`A`» type chip to «SELECT» command of ISO/IEC 14443-3 protocol (*Select Acknowledge*, *SAK*) – for the internal use by the main control library

`ATQ_B` – reply of the «`B`» type chip to the identification request (*Answer To Request*, *Type B – ATQB*) – for the internal use by the main control library

`BitRateS` – **`eRFID_BaudRate`** value, indicating the established rate for data transmitting to the RFID-chip

`BitRateR` – **`eRFID_BaudRate`** value, indicating the established rate for data receiving from the RFID-chip

6.3.14. TRF_EFCOM

`TRF_EFCOM` structure is used to store information about presence of informational data groups in the memory of RFID-chip supporting ISO/IEC 14443-4 protocol (see sections 5.7.4, 5.8.10).

```
struct TRF_EFCOM
{
  BYTE bLDSVersion[4];
  BYTE bUCVersion[6];
  BYTE bSizeDataGroup;
  BYTE bDataGroup[20];
```

```
  WORD wSizeGroup[20];
  WORD time[20];
};
```

Declaration: `RFID.h`

Fields:

`bLDSVersion` – version of logical data structure in `aabb` format, where `aa` – major version number, `bb` – minor version number

`bUCVersion` – Unicode version in `aabbcc` format, where `aa` – major version number, `bb` – minor version number, `cc` – release version number

`bSizeDataGroup` – number of significant elements in `bDataGroup`, `wAddrGroup` and `wSizeGroup`

`bDataGroup` – list of identifiers of the informational data groups that are present in the memory of the chip (a set of **`eRFID_DataGroupTypeTag`** values in the range from `RFDGT_DG1` to `RFDGT_SOD`)

`wSizeGroup` – lengths of corresponding informational data groups from `bDataGroup` list

`time` – time of reading of corresponding information data groups from `bDataGroup` list, ms

XML-representation of the structure:

```
<RFID_EF_COM>
  <LDSVersion>        – string contents of bLDSVersion
  <UnicodeVersion>    – string contents of bUCVersion
  <DataGroup Time="" Size="" Tag="">
                      – for each informational data group registered in EF.COM.
                        Time – reading time in ms, Size – size in bytes, Tag –
                        corresponding bDataGroup element.
</RFID_EF_COM>
```

6.3.15. TRF_FT_STRING

`TRF_FT_STRING` structure is used to store information about the text field that is a part of one of the informational data groups.

```
struct TRF_FT_STRING
{
  int       nType;
  DWORD     nStatus;
  BYTE      sFormat[32];
  unsigned  nDataLength;
  BYTE *    pData;
};
```

Declaration: `RFID.h`

Fields:

`nType`	–	logical type of the field (one of **`eVisualFieldType`** or **`eRFID_VisualFieldType`** values)
`nStatus`	–	result of logical analysis of compliance of the contents of the field with the requirements of the specification (`errLDS_Ok` or one of **`eLDS_ParsingNotificationCodes`** values) (see section 5.2)
`sFormat`	–	mask of format of text information (for example, «`YYMMDD`» for date of birth)
`nDataLength`	–	length of `pData` text array
`pData`	–	text array of the field contents

In XML-structures appears as a separate node, named based on the context of use. As the value the contents of `pData` in a string format appears. It is also possible the context presence of `Status`, `Type` and `Format` attributes of the node. As their contents are the values of the fields `nStatus`, `nType` and `sFormat` respectively.

6.3.16. TRF_FT_BYTE

`TRF_FT_BYTE` structure is used to store information about the numeric `BYTE` field (1 byte) that is a part of one of the informational data groups.

```
struct TRF_FT_BYTE
{
   int     nType;
   DWORD   nStatus;
   BYTE    bData;
};
```

Declaration: `RFID.h`

Fields:

`nType`	–	logical type of the field (one of **`eVisualFieldType`** or **`eRFID_VisualFieldType`** values)
`nStatus`	–	result of logical analysis of compliance of the contents of the field with the requirements of the specification (`errLDS_Ok` or one of **`eLDS_ParsingNotificationCodes`** values) (see section 5.2)
`bData`	–	numeric value

In XML-structures appears as a separate node, named based on the context of use. As the value the contents of `pData` in a numeric format appears. It is also possible the context presence of `Status` and `Type` attributes of the node. As their contents are the values of the fields `nStatus` and `nType` respectively.

6.3.17. TRF_FT_WORD

TRF_FT_WORD structure is used to store information about the numeric WORD field (2 bytes) that is a part of one of the informational data groups.

```
struct TRF_FT_WORD
{
  int    nType;
  DWORD  nStatus;
  WORD   wData;
};
```

Declaration: RFID.h

Fields:

nType – logical type of the field (one of **eVisualFieldType** or **eRFID_VisualFieldType** values)

nStatus – result of logical analysis of compliance of the contents of the field with the requirements of the specification (errLDS_Ok or one of **eLDS_ParsingNotificationCodes** values) (see section 5.2)

wData – numeric value

In XML-structures appears as a separate node, named based on the context of use. As the value the contents of pData in a numeric format appears. It is also possible the context presence of Status and Type attributes of the node. As their contents are the values of the fields nStatus and nType respectively.

6.3.18. TRF_FT_NUMBER

TRF_FT_NUMBER structure is used to store information about the numeric DWORD field (4 bytes) that is a part of one of the informational data groups.

```
struct TRF_FT_NUMBER
{
  int    nType;
  DWORD  nStatus;
  int    nData;
};
```

Declaration: RFID.h

Fields:

nType – logical type of the field (one of **eVisualFieldType** or **eRFID_VisualFieldType** values)

nStatus – result of logical analysis of compliance of the contents of the field with the requirements of the specification (errLDS_Ok

		or one of **eLDS_ParsingNotificationCodes** values) (see section 5.2)
nData	–	numeric value

In XML-structures appears as a separate node, named based on the context of use. As the value the contents of pData in a numeric format appears. It is also possible the context presence of Status and Type attributes of the node. As their contents are the values of the fields nStatus and nType respectively.

6.3.19. TRF_FT_BYTES

TRF_FT_BYTES structure is used to store an array of binary informationthat is a part of one of the informational data groups.

```
struct TRF_FT_BYTES
{
   int      nType;
   DWORD    nStatus;
   DWORD    nDataLength;
   BYTE    *pData;
};
```

Declaration:		RFID.h
Field:		
nType	–	logical type of the field (one of **VisualFieldType**, **eRFID_VisualFieldType** or **eGraphicFieldType** values)
nStatus	–	result of logical analysis of compliance of the contents of the field with the requirements of the specification (errLDS_Ok or one of **eLDS_ParsingNotificationCodes** values) (see section 5.2)
nDataLength	–	length of pData array
pData		binary data array

In XML-structures appears as a separate node, named based on the context of use. As the value the Base64-encoded contents of pData in CDATA-item format appears. It is also possible the context presence of Status, Type and Length attributes of the node. As their contents are the values of the fields nStatus, nType and nDataLength respectively.

6.3.20. TRF_EF_DG1

TRF_EF_DG1 structure used to store the contents of EF.DG1 informational data group of *ePassport* application – document MRZ data.

```
struct TRF_EF_DG1
{
  BYTE          nType;
  BYTE          nDocumentID;
  TRF_FT_STRING ftsDocumentType;
  TRF_FT_STRING ftsState;
  TRF_FT_STRING ftsHolder;
  TRF_FT_STRING ftsDocumentNumber;
  BYTE          nCheckDigitDocumentNumber;
  TRF_FT_STRING ftsNationality;
  TRF_FT_STRING ftsBirthday;
  BYTE          nCheckDigitBirthday;
  TRF_FT_STRING ftsSex;
  TRF_FT_STRING ftsExpiryDate;
  BYTE          nCheckDigitExpiryDate;
  TRF_FT_STRING ftsOptionalData;
  BYTE          nCheckDigitOptionalData;
  BYTE          nCheckDigitComposite;
};
```

Declaration: `RFID.h`

Field:

`nType`	– type of informational data group; always contains `RFDGT_DG1` value from **`eRFID_DataGroupTypeTag`** enumeration
`nDocumentID`	– type of document, one of **`CDocFormat`** values (classification of document formats – by the ISO/IEC 7810)
`ftsDocumentType`	– symbolic code of document type
`ftsState`	– symbolic code of document issuing state
`ftsHolder`	– DO’s name and surname of the
`ftsDocumentNumber`	– document number
`nCheckDigitDocumentNumber`	– check digit of document number
`ftsNationality`	– symbolic code of DO’s nationality
`ftsBirthday`	– DO’s date of birth
`nCheckDigitBirthday`	– check digit of DO’s date of birth
`ftsSex`	– DO’s sex
`ftsExpiryDate`	– term of validity of the document
`nCheckDigitExpiryDate`	– check digit of term of validity of the document
`ftsOptionalData`	– DO’s personal number or other additional data
`nCheckDigitOptionalData`	– check digit of additional data
`nCheckDigitComposite`	– general check digit

XML-representation of the structure:

```
<RFID_DG1>
  <Type>
  <DocumentID>
  <DocumentType>
```

```
    <State>
    <Holder>
    <DocumentNumber>
    <CheckDigitDocumentNumber>
    <Nationality>
    <Birthday>
    <CheckDigitBirthday>
    <Sex>
    <ExpiryDate>
    <CheckDigitExpiryDate>
    <OptionalData>
    <CheckDigitOptionalData>
    <CheckDigitComposite>
  </RFID_DG1>
```

Values and format of nodes correspond to the fields of **TRF_EF_DG1** structure.

6.3.21. TRF_EF_DG234

TRF_EF_DG234 structure is used to store the contents of EF.DG2, EF.DG3, EF.DG4 informational data groups of *ePassport* application and EF.DG6, EF.DG7, EF.DG8 informational data groups of *eDL* application – document owner's biometric data.

```
struct TRF_EF_DG234
{
  BYTE         nType;
  BYTE         nRecords;
  TRF_EF_BIT**   pBITs;
};
```

Declaration: RFID.h

Fields:

nType	–	type of informational data group; contains RFDGT_DG2, RFDGT_DG3, RFDGT_DG4, RFDGT_EDL_DG6, RFDGT_EDL_DG7 or RFDGT_EDL_DG8 values respectively (from **eRFID_DataGroupTypeTag** enumeration)
nRecords	–	number of pBITs array elements
pBITs	–	array of pointers to the container structures of records, included in the informational data group and containing templates of biometric information (*Biometric Information Template*, *BIT*)

XML-representation of the structure:

```
  <RFID_DG2>
    <Type>              – numeric nType value
    <BiometricTemplates>
```

```
      <BIT>             – XML-representation of TRF_EF_BIT structure, for each
                          element of pBITs
    </BiometricTemplates>
  </RFID_DG2>
```

Name of the root node corresponds to the informational data group.

6.3.22. TRF_EF_BIT

TRF_EF_BIT structure is a container for storing the contents of a single record (*Biometric Information Template*, *BIT*) of the biometric informational data group. It includes the contents of the corresponding *Biometric Header Template* (*BHT*) and directly the object of the biometric data themselves.

Biometric information is represented in a format that meets the requirements of [17] (ISO/IEC 19785.2), and corresponds to the structure of objects with biometric information, which is given in the Table D.2 of this document.

```
struct TRF_EF_BIT
{
  TRF_FT_NUMBER ftnSecurity;
  TRF_FT_NUMBER ftnIntegrity;
  TRF_FT_NUMBER ftnVersion;
  TRF_FT_NUMBER ftnType;
  TRF_FT_NUMBER ftnSubType;
  TRF_FT_STRING ftsCreateDate;
  TRF_FT_STRING ftsValidityPeriod;
  TRF_FT_NUMBER ftnProductID;
  TRF_FT_NUMBER ftnFormatOwner;
  TRF_FT_NUMBER ftnFormatType;
  TRF_FT_NUMBER ftnBDBType;
  TRF_FT_BYTES  ftbBDBFormatId;
  TRF_FT_BYTES  ftbBDBVersion;
  TRF_FT_BYTES  ftbRawBDBData;
  void *        pBDB;
};
```

Declaration: `RFID.h`

Fields:

`ftnSecurity` – one of **eBIT_SecurityOptions** values [17, § 5.2.1.1]

`ftnIntegrity` – one of **eBIT_IntegrityOptions** values [17, § 5.2.1.2]

`ftnVersion` – header version (*Patron Header Version*) [17, § 5.2.1.4]

`ftnType` – type of biometric data, one of **eCBEFF_BiometricType** values [17, § 5.2.1.5]

`ftnSubType` – subtype of biometric data [17, § 5.2.1.6] (combination **eCBEFF_BiometricSubTypeMask** values)

`ftsCreateDate` – date of creation of this biometric record [17, § 5.2.1.10]

`ftsValidityPeriod` – expiration date of this biometric record [17, § 5.2.1.11]

ftnProductID	–	product identifier (*Product ID*) [17, § 5.2.1.18]
ftnFormatOwner	–	identifier of the owner of biometric data representation format (one of **eCBEFF_FormatOwners** values) [17, § 5.2.1.17.1]
ftnFormatType	–	identifier of biometric data record format (one of **eCBEFF_FormatTypes** values) [17, § 5.2.1.17.2]
ftnBDBType	–	tag of biometric data group, value 0x5F2E or 0x7F2E [17, Table D.2]
ftbBDBFormatId	–	format identifier in a text form, contains symbol signature with a terminating zero byte
ftbBDBVersion	–	text representation of the format version with a terminating zero byte. The first two characters represent the major version number, the third one – the minor version number
ftbRawBDBData	–	complete binary image of the biometric record
pBDB	–	pointer to a structure containing the data of biometric record

Type of structure, referenced to by pBDB, is defined by the contents of ftnFormatType.

Field **ftnFormatType** value	Pointer type in **pBDB**
ftypFace_Image ftypFace_Image_FDIS	TFacialBDB *
ftypFinger_Image ftypFinger_Image_FDIS	TFingerBDB *
ftypIris_Image_FDIS ftypIris_ImageExtended ftypIris_ImageExtended_FDIS	TIrisBDB *
ftypFinger_Minutiae ftypFinger_Minutiae_FDIS ftypFinger_MinutiaeExtended ftypFinger_MinutiaeExtended_FDIS	TFingerMinutiaeBDB *

The contents of biometric data of other types will only be present in ftbRawBDBData binary non-formatted array.

Note. When **TRF_EF_BIT** structures used for description of biometric data, the original type of BDB format, which is directly specified in the read data, is stored in ftnFormatType field. However, this value due to some reason may not comply with the requirements of the specification. In this case the working value of BDB format type corrected by a number of other means is stored in **ftbRawBDBData.nType** field.

Belonging to a particular subtype of biometric data is determined by applying the appropriate mask from **eCBEFF_BiometricSubTypeMask** to the contents of ftnSubType field:

```
bool subtype = (ftnSubType.nData & <Mask>) == <Mask>;
```

XML-representation of the structure:

```
<BIT>
  <Security>
  <Integrity>
  <Version>
  <Type>
  <SubType>
  <CreateDate>
  <ValidityPeriod>
  <ProductID>
  <FormatOwner>
  <FormatType>
  <BDBType>
  <BDBFormatId>
  <BDBVersion>
  <node of corresponding BDB XML-representation>
</BIT>
```

Values and format of nodes correspond to the fields of **TRF_EF_BIT** structure.

6.3.23. TFacialBDB

TFacialBDB structure is used to store the contents of a single record with biometric graphic facial data according to [15].

```
struct TFacialBDB
{
  WORD              FacesCount;
  TFacialRecord *   pFaces;
};
```

Declaration: RFID.h

Fields:

FacesCount	–	number of pFaces array elements
pFaces	–	array of objects with biometric information included in the present record
ftbBDBFormatId	–	field of the respective **TRF_EF_BIT** record object must contain the value 'FAC'.

XML-representation of the structure:

<FacialBDB>		
<FacialRecord>	–	XML-representation of **TFacialRecord** structure, for each element of pFaces
</FacialBDB>		

6.3.24. TFacialRecord

TFacialRecord structure is used to store information about a single object from a record with biometric graphic facial data.

```
struct TFacialRecord
{
   TFacialInfo      frFacialInfo;
   TFacialImageInfo frImageInfo;
};
```

Declaration: `RFID.h`

Field:

`frFacialInfo` – general information about the object

`frImageInfo` – graphic and information about it

XML-representation of the structure:

`<FacialRecord>`

`<FacialInfo>` – XML-representation of **`TFacialInfo`** structure

`<ImageInfo>` – XML-representation of **`TFacialImageInfo`** structure

`</FacialRecord>`

6.3.25. TFacialInfo

`TFacialInfo` structure is used to store general information about a single object from a record with biometric graphic facial data.

```
struct TFacialInfo
{
   TRF_FT_BYTE       Gender;
   TRF_FT_BYTE       EyeColor;
   TRF_FT_BYTE       HairColor;
   TRF_FT_NUMBER     FeatureMask;
   TRF_FT_WORD       Expression;
   TPoseAngle        PoseAngle;
   TPoseAngle        PoseAngleUncertainty;
   WORD              FeatPointsCount;
   TFeaturePoint *   pFeaturePointsList;
};
```

Declaration: `RFID.h`

Fields:

`Gender` – DO's sex, one of **`eCBEFF_Gender`** values [15, § 5.5.3]

`EyeColor` – DO's eye color, one of **`eCBEFF_EyeColor`** values [15, § 5.5.4]

`HairColor` – DO's hair color, one of **`eCBEFF_HairColor`** values [15, § 5.5.5]

`FeatureMask` – byte combination of flags of presence of various face image features (combination of **`eCBEFF_FaceFeatureMask`** values) [15, § 5.5.6]

`Expression` – face expression on the image, one of **`eCBEFF_FaceExpression`** values [15, § 5.5.7]

`PoseAngle` – description of face pose [15, § 5.5.8]
`PoseAngleUncertainty` – description of face uncertainty pose [15, § 5.5.9]
`FeatPointsCount` – number of `pFeaturePointsList` array elements
`pFeaturePointsList` – array of feature points [15, § 5.6]

The presence of a particular feature of the face image is determined by applying the appropriate mask from **`eCBEFF_FaceFeatureMask`** to the contents of `FeatureMask` field:

```
bool feature = (FeatureMask & <Mask>) == <Mask>;
```

XML-representation of the structure:

```
<FacialInfo>
  <Gender>
  <EyeColor>
  <HairColor>
  <FeatureMask>
  <Expression>
  <PoseAngle>
  <PoseAngleUncertainty>
  <FeaturePoints>
      <FeaturePoint>
      ...
  </FeaturePoints>
</FacialInfo>
```

Values and format of nodes correspond to the fields of **`TFacialInfo`** structure.

6.3.26. TPoseAngle

`TPoseAngle` structure is used to store information about the face pose, contained in the object description in the record with biometric graphic data, according to [15, §5.5.8].

```
struct TPoseAngle
{
  TRF_FT_BYTE   Yaw;
  TRF_FT_BYTE   Pitch;
  TRF_FT_BYTE   Roll;
};
```

Declaration: `RFID.h`
Fields:
`Yaw` – yaw angle;
`Pitch` – pitch angle;
`Roll` – roll angle.

XML-representation of the structure:

```
<PoseAngle>
  <Yaw>
  <Pitch>
```

```
    <Roll>
  </PoseAngle>
```

Values and format of nodes correspond to the fields of **`TPoseAngle`** structure.

6.3.27. TFeaturePoint

`TFeaturePoint` structure is used to store information about a single face feature point contained in the object description in the record with biometic graphic data, according to [15, §5.6].

```
struct TFeaturePoint
{
  TRF_FT_BYTE  FeatureType;
  TRF_FT_BYTE  FeaturePoint;
  TRF_FT_WORD  X;
  TRF_FT_WORD  Y;
  TRF_FT_WORD  reserved;
};
```

Declaration: `RFID.h`

Fields:

`FeatureType`	–	type of the feature (always contains the value `1`)
`FeaturePoint`	–	code of the feature point (according to ISO/IEC 14496-2:2003)
`X`	–	X-coordinate of the point, relative to the upper left corner of the image
`Y`	–	Y-coordinate of the point, relative to the upper left corner of the image
`reserved`	–	reserved for further use

XML-representation of the structure:

```
  <FeaturePoint>
    <Type>
    <Point>
    <X>
    <Y>
  </FeaturePoint>
```

Values and format of nodes correspond to the fields of **`TFeaturePoint`** structure.

6.3.28. TFacialImageInfo

`TFacialImageInfo` structure is used to store a graphic image of the object from a record with biometric graphic facial data according to [15, §5.7].

```
struct TFacialImageInfo
{
  TRF_FT_BYTE   FaceImageType;
```

```
   TRF_FT_BYTE    ImageDataType;
   TRF_FT_WORD    Width;
   TRF_FT_WORD    Height;
   TRF_FT_BYTE    ImageColorSpace;
   TRF_FT_BYTE    SourceType;
   TRF_FT_WORD    DeviceType;
   TRF_FT_WORD    Quality;
   DWORD          ImageDataLength;
   BYTE           *pData;
};
```

Declaration: `RFID.h`

Fields:

`FaceImageType`	–	type of face image, one of **eCBEFF_FaceImageType** or **eCBEFF_FaceImageTypeFDIS** values [15, § 5.7.1]
`ImageDataType`	–	format of storing image data, one of **eCBEFF_ImageDataType** values [15, § 5.7.2]
`Width`	–	image width [15, § 5.7.3]
`Height`	–	image height [15, § 5.7.4]
`ImageColorSpace`	–	image color space, one of **eCBEFF_ImageColorSpace** values [15, § 5.7.5]
`SourceType`	–	image acquisition source, one of **eCBEFF_ImageSourceType** values [15, § 5.7.6]
`DeviceType`	–	identifier of the device by which the image was acquired (is defined by the manufacturer) [15, § 5.7.7]
`Quality`	–	image quality (reserved) [15, § 5.7.8]
`ImageDataLength`	–	size of `pData` array
`pData`	–	binary array representing the image in a format defined by the value of `ImageDataType` field – JPEG or JPEG2000

XML-representation of the structure:

```
<ImageInfo>
  <FaceImageType>
  <ImageDataType>
  <Width>
  <Height>
  <ImageColorSpace>
  <SourceType>
  <DeviceType>
  <Quality>
  <Data Length="">
      <![CDATA[]]>
  </Data>
</ImageInfo>
```

Values and format of nodes correspond to the fields of **TFacialImageInfo**. `Length` attribute of `Data` node contains the value of `ImageDataLength` field, node content – Base64-encoded data from `pData`.

6.3.29. TFingerBDB

TFingerBDB structure is used to store the contents of a single record with graphic data of fingerprints (palms) [14]. ftbBDBFormatId field of the respective **TRF_EF_BIT** record object must contain the value 'FIR' [14, §7.1.1].

```
struct TFingerBDB
{
   TRF_FT_WORD    CaptureDeviceID;
   TRF_FT_WORD    ImageAcquisitionLevel;
   TRF_FT_BYTE    ScaleUnits;
   TRF_FT_WORD    ScanResolutionHorz;
   TRF_FT_WORD    ScanResolutionVert;
   TRF_FT_WORD    ImageResolutionHorz;
   TRF_FT_WORD    ImageResolutionVert;
   TRF_FT_BYTE    PixelDepth;
   TRF_FT_BYTE    ImageCompressionAlgorithm;
   TRF_FT_WORD    reserved;
   BYTE           FingersCount;
   TFingerRecord *pFingers;
};
```

Declaration: RFID.h

Fields:

CaptureDeviceID	–	identifier of the device by which the templates of biometric information were acquired (is defined by the manufacturer) [14, § 7.1.4]
ImageAcquisitionLevel	–	image acquisition level [14, § 7.1.5, Table 1]
ScaleUnits	–	image resolution units, one of **eCBEFF_ScaleUnits** values [14, § 7.1.7]
ScanResolutionHorz	–	horizontal resolution of the scanning device in ScaleUnits units [14, § 7.1.8]
ScanResolutionVert	–	vertical resolution of the scanning device in ScaleUnits units [14, § 7.1.9]
ImageResolutionHorz	–	horizontal resolution of the images in ScaleUnits units [14, § 7.1.10]
ImageResolutionVert	–	vertical resolution of the images in ScaleUnits units [14, § 7.1.11]
PixelDepth	–	number of bits per color used in the images [14, § 7.1.12]
ImageCompressionAlgorithm	–	identifier of image compression algorithm, one of **eCBEFF_ImageCompressionAlgorithm** values [14, § 7.1.13]
reserved	–	reserved [14, § 7.1.14]
FingersCount	–	number of pFingers array elements

`pFingers` – array of templates of fingerprints (palms), included in the current biometric record.

XML-representation of the structure:

```
<FingerBDB>
  <CaptureDeviceID>
  <ImageAcquisitionLevel>
  <ScaleUnits>
  <ScanResolutionHorz>
  <ScanResolutionVert>
  <ImageResolutionHorz>
  <ImageResolutionVert>
  <PixelDepth>
  <ImageCompressionAlgorithm>
  <FingerRecord>
  ...
</FingerBDB>
```

Values and format of nodes correspond to the fields of **`TFingerBDB`** structure. The number of `FingerRecord` elements corresponds to the number of elements in `pFingers` array.

6.3.30. TFingerRecord

`TFingerRecord` structure is used to store data of a single template of fingerprints (palm) from the record with biometric graphic data.

```
struct TFingerRecord
{
  TRF_FT_BYTE    Position;
  TRF_FT_BYTE    ImageQuality;
  TRF_FT_BYTE    ImpressionType;
  TRF_FT_WORD    HorzLineLength;
  TRF_FT_WORD    VertLineLength;
  BYTE           ViewsCount;
  DWORD          *ViewDataLength;
  BYTE           **pViewData;
};
```

Declaration: `RFID.h`

Fields:

`Position` – position of a finger (palm), one of **`eC-BEFF_FingerPalmPosition`** values [14, § 7.2.2]

`ImageQuality` – image quality [14, § 7.2.5]

`ImpressionType` method of receiving a fingerprint, one of **`eC-BEFF_FingerPalmImpression`** values [14, § 7.2.6]

`HorzLineLength` – image width, in pixels [14, § 7.2.7]

`VertLineLength` – image height, in pixels [14, § 7.2.8]

`ViewsCount` – number of `ViewDataLength` and `pViewData` arrays elements

`ViewDataLength` – sizes of the respective binary arrays, pointers to which are contained in `pViewData`

`pViewData` – array of pointers to binary arrays containing graphic images of variants of the current fingerprint template. The data format of these arrays is defined by the value of `ImageCompressionAlgorithm` field of the parent **TFingerBDB** structure

XML-representation of the structure:

```
<FingerRecord>
  <Position>
  <ImageQuality>
  <ImpressionType>
  <HorzLineLength>
  <VertLineLength>
  <Views>
      <ViewData Length="">
         <![CDATA[]]>
      </ViewData>
      ...
  </Views>
</FingerRecord>
```

Values and format of nodes correspond to the fields of **TFingerRecord** structure. The number of `ViewData` elements in `Views` corresponds to the number of elements in `pViewData` array. `Length` attribute of `ViewData` node contains the value of the corresponding element of `ViewDataLength` array, node content – Base64-encoded `pViewData` data.

6.3.31. TFingerMinutiaeBDB

TFingerMinutiaeBDB structure is used to store the contents of a single record with data of the encoded fingerprint according to [13]. `ftbBDBFormatId` field of the respective **TRF_EF_BIT** record object must contain the value ‘`FMR`’ [13, §7.3.1].

```
struct TFingerMinutiaeBDB
{
  TRF_FT_WORD          CaptureDeviceID;
  TRF_FT_WORD          ImageWidth;
  TRF_FT_WORD          ImageHeight;
  TRF_FT_WORD          ResolutionX;
  TRF_FT_WORD          ResolutionY;
  long                 ViewsCount;
  TFingerMinutiaeRecord *pFingers;
};
```

Declaration: RFID.h

Fields:

CaptureDeviceID – identifier of the device by which the templates of biometric information were acquired (is defined by the manufacturer) [13, § 7.3.4, § 7.3.5]

ImageWidth – original fingerprint image width, in pixels [13, § 7.3.6]

ImageHeight – original fingerprint image height, in pixels [13, § 7.3.7]

ResolutionX – resolution of minutiae system on X, pixels/cm [13, § 7.3.6]

ResolutionY – resolution of minutiae system on Y, pixels/cm [13, § 7.3.6]

ViewsCount – number of pFingers array elements

pFingers – array of structures containing information about the encoded variants of fingerprints included in the current record of the biometric information

XML-representation of the structure:

```
<FingerMinutiaeBDB>
  <CaptureDeviceID>
  <ImageWidth>
  <ImageHeight>
  <ResolutionX>
  <ResolutionY>
  <FingerMinutiaeRecord>
  ...
</FingerMinutiaeBDB>
```

Values and format of nodes correspond to the fields of **TFingerMinutiaeBDB** structure. The number of FingerMinutiaeRecord elements corresponds to the number of elements in pFingers array.

6.3.32. TFingerMinutiaeRecord

TFingerMinutiaeRecord structure is used to store data of a single encoded variant of fingerprint in the biometric data record.

```
struct TFingerMinutiaeRecord
{
  long             FingerPosition;
  long             ViewNumber;
  long             Impression;
  long             Quality;
  long             MinutiaeCount;
  TOneMinutia      *pMinutiae;
  long             ExtendedDataCount;
  TMinutiaeExtData *pExtendedData;
};
```

Declaration: RFID.h

Fields:

`FingerPosition`	–	finger position, one of **`eCBEFF_FingerPalmPosition`** values [13, § 7.4.1.1]
`ViewNumber`	–	number of the variant of fingerprint [13, § 7.4.1.1]
`Impression`	–	method of fingerprint acquisition, one of **`eC-BEFF_FingerPalmImpression`** values [13, § 7.4.1.3]
`Quality`	–	average quality of minutiae data [13, § 7.4.1.4]
`MinutiaeCount`	–	number of `pMinutiae` array elements
`pMinutiae`	–	array of pointers to structures containing minutiae information, encoding the current print variant [13, § 7.4.2]
`ExtendedDataCount`	–	number of `pExtendedData` array elements
`pExtendedData`	–	array of structures containing additional information about the current encoded fingerprint variant [13, § 7.5]

XML-representation of the structure:

```
<FingerMinutiaeRecord>
  <FingerPosition>
  <ViewNumber>
  <Impression>
  <Quality>
  <Minutiae>
      <Minutia>
      ...
  </Minutiae>
  <ExtendedData>
      <MinutiaeExtData>
      ...
  </ExtendedData>
</FingerMinutiaeRecord>
```

Values and format of nodes correspond to the fields of **`TFingerMinutiaeRecord`** structure. The number of `Minutiae` elements corresponds to the number of elements in `pMinutiae` array, the number of `ExtendedData` elements – to `pExtendedData` array.

6.3.33. TOneMinutia

`TOneMinutia` structure is used to store information about a single minutia.

```
struct TOneMinutia
{
  long   Type;
  long   XPos;
  long   YPos;
  long   Angle;
  long   Quality;
};
```

Declaration: `RFID.h`

Fields:

`Type`	–	minutia type [13, § 7.4.2.1]
`XPos`	–	`X`-coordinate of minutia [13, § 7.4.2.2]
`YPos`	–	`Y`- coordinate of minutia [13, § 7.4.2.2]
`Angle`	–	angle of minutia position [13, § 7.4.2.3]
`Quality`	–	minutia quality [13, § 7.4.2.4]

XML-representation of the structure:

```
<Minutia>
  <Type>
  <XPos>
  <YPos>
  <Angle>
  <Quality>
</Minutia>
```

Values and format of nodes correspond to the fields of **`TOneMinutia`** structure.

6.3.34. TMinutiaeExtData

`TMinutiaeExtData` structure is used to store additional information about the encoded fingerprint.

```
struct TMinutiaeExtData
{
  WORD DataType;
  void *pData;
};
```

Declaration: `RFID.h`

Field:

`DataType`	–	type of additional data stored in `pData` (one of `eMinutiaeExtendedDataType` values)
`pData`	–	pointer to a structure of the additional data

Type of structure pointed to by `pData`, is defined by the contents of `DataType`:

- if `DataType` equal to **`medtRidgeCountData`**, `pData` contains a pointer to **`TMinutiaeRidgeCountData`** structure;
- if `DataType` equal to **`medtCoreAndDeltaData`**, `pData` contains a pointer to **`TCoreAndDeltaData`** structure;
- if `DataType` equal to **`medtZonalQualityData`**, `pData` contains a pointer to **`TZonalQualityData`** structure.

XML-representation of the structure:

```
  <MinutiaeExtData>
    <DataType>
    <node of data structure XML-representation>
  </MinutiaeExtData>
```

Values and format of nodes correspond to the fields of **TMinutiaeExtData** structure.

6.3.35. TMinutiaeRidgeCountData

TMinutiaeRidgeCountData structure is used to store additional information about the encoded fingerprint – number of ridges between the pairs of minutiae [13, §7.5.2].

```
struct TMinutiaeRidgeCountData
{
   long              ExtractionMethod;
   long              DataCount;
   TRidgeCountData  *RidgeCountData;
};
```

Declaration: `RFID.h`

Fields:

`ExtractionMethod`	–	method of ridge detection, one of **eRidgeCountExtractionMethod** values
`DataCount`	–	number of `RidgeCountData` array elements
`RidgeCountData`	–	array of structures of data on the number of ridges between the pairs of minutiae

XML-representation of the structure:

```
  <MinutiaeRidgeCountData>
    <ExtractionMethod>
    <RidgeCountData>
        <RidgeCountDataItem>
        ...
    </RidgeCountData>
  </MinutiaeRidgeCountData>
```

Values and format of nodes correspond to the fields of **TMinutiaeRidgeCountData** structure. The number of `RidgeCountData` elements corresponds to the number of elements in `RidgeCountData` array.

6.3.36. TRidgeCountData

TRidgeCountData structure is used to store information about the number of ridges located between the pair of minutiae, in additional data of the encoded fingerprint.

```
struct TRidgeCountData
{
```

```
   BYTE MinutiaeIndex1;
   BYTE MinutiaeIndex2;
   BYTE Count;
};
```

Declaration: `RFID.h`

Field:

`MinutiaeIndex1` – index of the first minutia of the given pair (in `pMinutiae` array of **`TFingerMinutiaeRecord`** structure)

`MinutiaeIndex2` – index of the second minutia of the given pair (in `pMinutiae` array of **`TFingerMinutiaeRecord`** structure)

`Count` – number of ridges located between the present pair of minutiae.

XML-representation of the structure:

```
<RidgeCountDataItem>
  <MinutiaeIndex1>
  <MinutiaeIndex2>
  <Count>
</RidgeCountDataItem>
```

Values and format of nodes correspond to the fields of **`TRidgeCountData`** structure.

6.3.37. TCoreAndDeltaData

`TCoreAndDeltaData` structure is used to store additional information on the encoded fingerprint – data on cores and deltas [13, §7.5.3].

```
struct TCoreAndDeltaData
{
   long        CoreAngleIsSpecified;
   long        CoresCount;
   TCoreData   *CoreData;
   long        DeltaAngleIsSpecified;
   long        DeltasCount;
   TDeltaData  *DeltaData;
};
```

Declaration: `RFID.h`

Fields:

`CoreAngleIsSpecified` – flag showing the presence of angular information on cores

`CoresCount` – number of `CoreData` array elements

`CoreData` – array of data on cores

`DeltaAngleIsSpecified` – flag showing the presence of angular information on deltas

`DeltasCount` – number of `DeltaData` array elements

`DeltaData` – array of data on deltas

XML-representation of the structure:

```
<CoreAndDeltaData>
  <CoreAngleIsSpecified>
  <DeltaAngleIsSpecified>
  <CoreData>
      <CoreDataItem>
      ...
  </CoreData>
  <DeltaData>
      <DeltaDataItem>
      ...
  </DeltaData>
</CoreAndDeltaData>
```

Values and format of nodes correspond to the fields of **`TCoreAndDeltaData`** structure. The number of `CoreData` elements corresponds to the number of elements in `CoreData` array, the number of `DeltaData` elements – to `DeltaData` array.

6.3.38. TCoreData

`TCoreData` structure is used to store information on a single core in the additional data of the encoded fingerprint.

```
struct TCoreData
{
  WORD X;
  WORD Y;
  BYTE Angle;
};
```

Declaration: `RFID.h`

Fields:

`X`	–	`X`-coordinate of the core point
`Y`	–	`Y`- coordinate of the core point
`Angle`	–	angle of the core point (in units of 1,40625 degrees). It has a value only in case of `CoreAngleIsSpecified=TRUE` in the parent **`TCoreAndDeltaData`** structure

XML-representation of the structure:

```
<CoreDataItem>
  <X>
  <Y>
  <Angle>
</CoreDataItem>
```

Values and format of nodes correspond to the fields of **`TCoreData`** structure.

6.3.39. TDeltaData

`TDeltaData` structure is used to store information about a single delta in the additional data of the encoded fingerprint.

```
struct TDeltaData
{
  WORD X;
  WORD Y;
  BYTE Angles[3];
};
```

Declaration: `RFID.h`

Field:

`X`	–	`X`- coordinate of the delta point
`Y`	–	`Y`- coordinate of the delta point
`Angles[3]`	–	angles of delta point (in units of 1,40625 degrees). It has a value only in case of `DeltaAngleIsSpecified=TRUE` in the parent **`TCoreAndDeltaData`** structure

XML-representation of the structure:

```
<DeltaDataItem>
  <X>
  <Y>
  <Angle1>
  <Angle2>
  <Angle3>
</DeltaDataItem>
```

Values and format of nodes correspond to the fields of **`TDeltaData`** structure.

6.3.40. TZonalQualityData

`TZonalQualityData` structure is used to store information about the fingerprint image quality map, divided into zones, in the additional data of the encoded fingerprint [13, §7.5.4].

```
struct TZonalQualityData
{
  BYTE CellWidth;
  BYTE CellHeight;
  BYTE CellInfoBitDepth;
  long CellDataLength;
  BYTE *CellQualityData;
};
```

Declaration: `RFID.h`

Fields:

`CellWidth` – grid spacing, dividing the image into zones horizontally
`CellHeight` – grid spacing, dividing the image into zones vertically
`CellInfoBitDepth` – number of bits representing the value of the quality of one zone
`CellDataLength` – length of `CellQualityData` array
`CellQualityData` – array of element of quality assessment of image zones

XML-representation of the structure:

```
<ZonalQualityData>
  <CellWidth>
  <CellHeight>
  <CellInfoBitDepth>
  <CellQualityData Length="">
      <![CDATA[]]>
  </CellQualityData>
</ZonalQualityData>
```

Values and format of nodes correspond to the fields of **TZonalQualityData** structure. `Length` attribute of `CellQualityData` node contains the value of `CellDataLength` field, node content – Base64-encoded `CellQualityData` data.

6.3.41. TIrisBDB

TIrisBDB structure is used to store the contents of a single record with iris graphic data according to [16]. `ftbBDBFormatId` field of the respective **TRF_EF_BIT** record object must contain the value '`IIR`'.

```
struct TIrisBDB
{
   TRF_FT_WORD    CaptureDeviceID;
   TRF_FT_WORD    ImageProperties;
   TRF_FT_WORD    IrisDiameter;
   TRF_FT_WORD    ImageFormat;
   TRF_FT_WORD    ImageWidth;
   TRF_FT_WORD    ImageHeight;
   TRF_FT_BYTE    IntensityDepth;
   TRF_FT_BYTE    ImageTransformation;
   BYTE           DUID[16];
   BYTE           EyesCount;
   TEyeRecord *   pEyes;
};
```

Declaration: `RFID.h`

Field:

`CaptureDeviceID` – identifier of the device by which the templates of biometric information were acquired (is defined by the manufacturer)
`ImageProperties` – properties of images, a combination of **eIrisImageProperties** values
`IrisDiameter` – expected diameter of iris, points

`ImageFormat`	–	identifier of image's format, one of **`eIrisImageFormat`** values
`ImageWidth`	–	width of images
`ImageHeight`	–	height of images
`IntensityDepth`	–	number of bits per color used in images
`ImageTransformation`	–	type of transformation to polar coordinates, one of **`eIrisImageTransformation`** values
`DUID`	–	unique identifier of the device by which the templates of biometric information were acquired
`EyesCount`	–	number of `pEyes` array elements
`pEyes`	–	array of biometric information templates

The presence of a particulat image property is determined by applying the appropriate mask from **`eIrisImageProperties`** to the contents of `ImageProperties` field, for example:

```
bool present = (value & iipmScanType_Mask) = iipmScanType_Corrected;
```

XML-representation of the structure:

```
<IrisBDB>
  <CaptureDeviceID>
  <ImageProperties>
  <IrisDiameter>
  <ImageFormat>
  <ImageWidth>
  <ImageHeight>
  <IntensityDepth>
  <ImageTransformation>
  <DUID>
  <EyeRecord>
  ...
</IrisBDB>
```

Values and format of nodes correspond to the fields of **`TIrisBDB`** structure. The number of `EyeRecord` elements corresponds to the number of elements in `pEyes` array. `DUID` node contains Base64-encoded data.

6.3.42. TEyeRecord

`TEyeRecord` structure is used to store data of a single template of iris image from the biometric graphic data record [16, § 5.5].

```
struct TEyeRecord
{
  TRF_FT_BYTE    BiometricSubtype;
  WORD           ImagesCnt;
  TIrisImage *   pImages;
};
```

Declaration: `RFID.h`

Field:

`BiometricSubtype` – type of template, one of **`eIrisSubtype`** values

`ImagesCnt` – number of `pImages` array elements

`pImages` – array of variants of the current biometric information template

XML-representation of the structure:

```
<EyeRecord>
  <BiometricSubtype>
  <IrisImage>
  ...
</EyeRecord>
```

Values and format of nodes correspond to the fields of **`TIrisBDB`** structure. The number of `IrisImage` elements corresponds to the number of elements in `pImages` array.

6.3.43. TIrisImage

`TIrisImage` structure is used to store a single variant of iris template from the biometric graphic data record [16, §5.5].

```
struct TIrisImage
{
  TRF_FT_WORD    ImageNumber;
  TRF_FT_BYTE    Quality;
  TRF_FT_WORD    RotationAngle;
  TRF_FT_WORD    RotationUncertainty;
  DWORD          DataLength;
  BYTE           *pData;
};
```

Declaration: `RFID.h`

Field:

`ImageNumber` – serial number of the template variant

`Quality` – image quality [16, § 5.5, Table A.1]

`RotationAngle` – rotation angle of iris image. The value calculated by the formula

```
(signed short) round(65536*angle/360) mod 65536
```

where `angle` – rotation angle, in degrees;

`RotationUncertainty` – allowed rotation angle of iris image. The value calculated by the formula

```
(signed short) round(65536*angle/180)
```

where `angle` – rotation angle, in degrees;

DataLength	–	size of pData array
pData	–	binary array containing graphic image of the current variant of iris template. The data format is defined by the contents of ImageFormat field of the parent **TIrisBDB** structure

XML-representation of the structure:

```
<IrisImage>
  <ImageNumber>
  <Quality>
  <RotationAngle>
  <RotationUncertainty>
  <Data Length="">
      <![CDATA[]]>
  </Data>
</ IrisImage>
```

Values and format of nodes correspond to the fields of **TIrisImage** structure. Length attribute of Data node contains the value of DataLength field, node content – Base64-encoded pData data.

6.3.44. TRF_EF_DG567

TRF_EF_DG567 structure is used to store the contents of EF.DG5, EF.DG6, EF.DG7 informational data groups of *ePassport* application and EF.DG5 of *eDL* application – additional graphic data.

```
struct TRF_EF_DG567
{
  BYTE          nType;
  BYTE          nRecords;
  TRF_FT_BYTES**pImages;
};
```

Declaration:		RFID.h
Fields:		
nType	–	type of informational data group; always contains RFDGT_DG5, RFDGT_DG6, RFDGT_DG7 or RFDGT_EDL_DG5 values from **eRFID_DataGroupTypeTag** enumeration respectively
nRecords	–	number of pImages array elements
pImages	–	array of pointers to container structures included in the structure of informational data group and containing graphic information

The binary data in pImages elements are the images of files containing the encoded graphics. For *ePassport* application, JPEG algorithm (ISO/IEC 10918) is applied for encoding of `EF.DG5` and `EF.DG7` groups; ANSI/NIST-ITL 1-2000 – for `EF.DG6` group. For *eDL* application, WSQ (IAFIS-IC-0110v3), JPEG (ISO/IEC 10918) or JPEG2000 (ISO/IEC 15444-1) formats can be used.

XML-representation of the structure:

```
<RFID_DG5>
  <Type>                  – numeric nType value
  <Images>
      <ImageRecord>       – XML-representation of TRF_EF_BYTES structure, for
                            each element of pImages array
  </Images>
</RFID_DG5>
```

Name of the root node corresponds to the informational data group.

6.3.45. TRF_EF_DG8910

TRF_EF_DG8910 structure is used to store the contents of `EF.DG8`, `EF.DG9` and `EF.DG10` informational data groups (*ePassport* application). Since the format of these data groups has not yet been defined in standard their contents are stored in as a set of standard binary arrays.

```
struct TRF_EF_DG8910
{
  BYTE            nType;
  BYTE            nRecords;
  TRF_FT_BYTES**pContents;
};
```

Declaration: `RFID.h`

Fields:

`nType`	–	type of informational data group; always contains `RFDGT_DG8`, `RFDGT_DG9` or `RFDGT_DG10` values from **eRFID_DataGroupTypeTag** enumeration respectively
`nRecords`	–	number of `pContents` array elements
`pContents`	–	array of pointers to binary data arrays

XML-representation of the structure:

```
<RFID_DG8>
  <Type>                  – numeric nType value
  <Contents>
      <Record>            – XML- representation of TRF_EF_BYTES structure, for
                            each element of pContents array
  </Contents>
</RFID_DG8>
```

Name of the root node corresponds to the informational data group.

6.3.46. TRF_EF_DG11

TRF_EF_DG11 structure is used to store the contents of EF.DG11 information group of additional personal data of the DO (*ePassport* application).

```
struct TRF_EF_DG11
{
   BYTE           nType;
   TRF_FT_STRING ftsFullName;
   TRF_FT_STRING ftsPersonalNumber;
   TRF_FT_STRING ftsBirthday;
   TRF_FT_STRING ftsBirthdayPlace;
   TRF_FT_STRING ftsPermanentAddress;
   TRF_FT_STRING ftsPhone;
   TRF_FT_STRING ftsProfession;
   TRF_FT_STRING ftsTitle;
   TRF_FT_STRING ftsPersonalSummary;
   TRF_FT_STRING ftsOtherID;
   TRF_FT_STRING ftsCustodyInfo;
   TRF_FT_BYTES  ftbProofOfCitizenship;
   BYTE           nNamesCount;
   TRF_FT_STRING ftsFullNamesAdditional[16];
};
```

Declaration: RFID.h

Fields:

nType	– type of informational data group; always contains RFDGT_DG11 value from **eRFID_DataGroupTypeTag** enumeration
ftsFullName	– DO's full name in national symbols (ICAO 9303)
ftsPersonalNumber	– personal number
ftsBirthday	– full date of birth (in "yyyymmdd" format, where yyyy – year, mm – month, dd – day)
ftsBirthdayPlace	– place of birth
ftsPermanentAddress	– permanent place of residence
ftsPhone	– phone number
ftsProfession	– profession
ftsTitle	– title
ftsPersonalSummary	– additional data
ftsOtherID	– other valid ID numbers (separated by the symbol '<')
ftsCustodyInfo	custody information
ftbProofOfCitizenship	– image of graphic file of document image proving any citizenship (JPEG by ISO/IEC 10918)
nNamesCount	– number of ftsFullNamesAdditional array elements
ftsFullNamesAdditional	– DO's other names (ICAO 9303)

XML-representation of the structure:

```
<RFID_DG11>
  <Type>
  <FullName>
  <PersonalNumber>
  <Birthday>
  <BirthdayPlace>
  <PermanentAddress>
  <Phone>
  <Profession>
  <Title>
  <PersonalSummary>
  <OtherID>
  <CustodyInfo>
  <ProofOfCitizenship>
</RFID_DG11>
```

Values and format of nodes correspond to the fields of **TRF_EF_DG11** structure.

6.3.47. TRF_EF_DG12

TRF_EF_DG12 structure is used to store the contents of EF.DG12 information group of additional information on the document (*ePassport* application).

```
struct TRF_EF_DG12
{
  BYTE          nType;
  TRF_FT_STRING ftsAuthority;
  TRF_FT_STRING ftsIssueDate;
  TRF_FT_STRING ftsObservation;
  TRF_FT_STRING ftsTax;
  TRF_FT_BYTES  ftbImageFront;
  TRF_FT_BYTES  ftbImageRear;
  TRF_FT_STRING ftsPersonalization;
  TRF_FT_STRING ftsSerialNumber;
  BYTE          nPersonsNumber;
  TRF_FT_STRING ftsPersonName[16];
};
```

Declaration: RFID.h

Fields:

nType – type of informational data group; always contains RFDGT_DG12 value from **eRFID_DataGroupTypeTag** enumeration

ftsAuthority – authority that has issued the document

ftsIssueDate – date of issue (in "yyyymmdd" format, where yyyy – year, mm – month, dd – day)

ftsObservation – observation

`ftsTax` – tax information

`ftbImageFront` – image of graphic file of document face side image (JPG by ISO/IEC 10918)

`ftbImageRear` – image of graphic file of document reverse side image (JPG by ISO/IEC 10918)

`ftsPersonalization` – date and time of document personalization (in "`yyymmddhhmmss`" format, where `yyyy` – year, `mm` – month, `dd` – day, `hh` – hour, `mm` – minute, `ss` – second)

`ftsSerialNumber` – serial number of personalization system

`nPersonsNumber` – number of `ftsPersonName` array elements

`ftsPersonName` – names of other persons mentioned in the document (ICAO 9303)

XML-representation of the structure:

```
<RFID_DG12>
  <Type>
  <Authority>
  <IssueDate>
  <Observation>
  <Tax>
  <ImageFront>
  <ImageRear>
  <Personalization>
  <SerialNumber>
</RFID_DG12>
```

Values and format of nodes correspond to the fields of **`TRF_EF_DG12`** structure.

6.3.48. TRF_EF_DG_BINARY_ARRAY

`TRF_EF_DG_BINARY_ARRAY` structure is used to store the binary representation of the contents of informational data group of *`ePassport`* application, use of which is restricted to the internal use of SDK (`DG14`, `DG15`, `EF.SOD`) or the format of which is private for organizations having issued the travel document (`DG13`).

```
struct TRF_EF_DG_BINARY_ARRAY
{
  BYTE          nType;
  TRF_FT_BYTES pContents;
};
```

Declaration: `RFID.h`

Field:

`nType` – type of informational data group; contains **`eRFID_DataGroupTypeTag`** values

`pContents` – binary array of the data group contents

XML-representation of the structure:

```
<RFID_DG13>
  <Type>
  <Contents>
</RFID_DG13>
```

Name of the root node corresponds to the informational data group. Values and format of nodes correspond to the fields of **TRF_EF_DG_BINARY_ARRAY** structure.

6.3.49. TRF_EF_DG16

TRF_EF_DG16 structure is used to store the contents of EF.DG16 data group with information on persons to notify in case of emergency (*ePassport* application).

```
struct TRF_EF_DG16
{
  BYTE              nType;
  BYTE              nRecords;
  TRF_EF_PERSON**   pPersons;
};
```

Declaration: RFID.h

Fields:

nType – type of informational data group; always contains RFDGT_DG16 value from **eRFID_DataGroupTypeTag** enumeration

nRecords – number of pPersons array elements

pPersons – array of pointers to structures with information about specific persons

XML-representation of the structure:

```
<RFID_DG16>
  <Type>
  <Persons>
     <PersonItem>
     ...
  </Persons>
</RFID_DG16>
```

Values and format of nodes correspond to the fields of **TRF_EF_DG16** structure. The number of Persons elements corresponds to the number of elements in pPersons array.

6.3.50. TRF_EF_PERSON

TRF_EF_PERSON structure is used to store information about the person to notify in case of emergency.

```
struct TRF_EF_PERSON
{
  TRF_FT_STRING ftsRecordDate;
  TRF_FT_STRING ftsName;
  TRF_FT_STRING ftsPhone;
  TRF_FT_STRING ftsAddress;
};
```

Declaration: `RFID.h`

Fields:

`ftsRecordDate`	–	date of record
`ftsName`	–	full name
`ftsPhone`	–	phone number
`ftsAddress`	–	address

XML-representation of the structure:

```
<PersonItem>
  <RecordDate>
  <Name>
  <Phone>
  <Address>
</PersonItem>
```

Values and format of nodes correspond to the fields of **TRF_EF_PERSON** structure.

6.3.51. TRF_Authentification

TRF_Authentification structure is used to store the results of performing different stages of data authentication when working with SDK in the batch mode (see section 5.7.7).

```
struct TRF_Authentification
{
  DWORD              SODErrorStatus;
  DWORD              nSODNotifications;
  DWORD              *SODNotifications;
  DWORD              version;
  TRF_FT_BYTES       SO_DigestAlgorithm;
  TRF_FT_BYTES       SO_ID;
  DWORD              nDataGroupDigests;
  TRF_SOD_DG_Digest  *DataGroupDigests;
  DWORD              nSignerInfos;
  TRF_SOD_SignerInfo **SignerInfos;
  TRF_FT_BYTES       AA_KeyAlgorithm;
  DWORD              AA_Status;
  TRF_FT_BYTES       CA_Scheme;
  TRF_FT_BYTES       CA_KeyAlgorithm;
  DWORD              CA_Status;
  TRF_FT_BYTES       TA_Scheme;
  DWORD              TA_Status;
```

```
};
```

Declaration: `RFID.h`

Fields:

`SODErrorStatus` – status of preliminary `EF.SOD` data group parse (one of **`eSOD_Error_Status`** values)

`nSODNotifications` – number of **`SODNotifications`** array elements

`SODNotifications` – array of the codes of notifications appeared during the process of preliminary `EF.SOD` data group parse (respective values from **`eLDS_ParsingErrorCodes`** or **`eLDS_ParsingNotificationCodes`** enumerations)

`version` – `EF.SOD` structure version [3, § C.1]

`SO_DigestAlgorithm` – algorithm identifier of informational data groups hashing [3, § 3.3.5, C.1]

`SO_ID` – `EF.SOD` structure identifier [3, § C.1, C.2]

`nDataGroupDigests` – number of **`DataGroupDigests`** array elements

`DataGroupDigests` – array of results of comparison of hash values computed by the data read from the RFID-chip, with values from `EF.SOD` (for each of the present informational data group)

`nSignerInfos` – number of **`SignerInfos`** array elements

`SignerInfos` – array of results of the authentication of digital signatures from `EF.SOD`

`AA_KeyAlgorithm` – algorithm identifier of active authentication key [3, § 3.4.2, D.1, D.2]

`AA_Status` – result of active authentication (a value from **`eLDS_ParsingErrorCodes`** or **`eLDS_ParsingNotificationCodes`**)

`CA_Scheme` – CA scheme algorithm identifier [1, § A.1.1.1, A.2]

`CA_KeyAlgorithm` – CA key algorithm identifier [1, § A.1.1.1]

`CA_Status` – result of CA (a value from **`eLDS_ParsingErrorCodes`** or **`eLDS_ParsingNotificationCodes`**)

`TA_Scheme` – TA scheme algorithm identifier [1, § A.1.1.2, A.3]

`TA_Status` – result of TA (a value from **`eLDS_ParsingErrorCodes`** or **`eLDS_ParsingNotificationCodes`**)

XML-representation of the structure:

```
<RFID_Authentication_Info>
  <SODErrorStatus>    – numeric SODErrorStatus value
  <SODNotifications>
     <SODNotification Value=""/>
     ...
  </SODNotifications>– numeric SODNotifications values (in Value attributes)
  <Version>           – numeric Version value
```

```
    <SO_DigestAlgorithm> – text SO_DigestAlgorithm value
    <SO_ID>              – text SO_ID value
    <DataGroupDigests>
       <DataGroupDigest>
       ...
    </DataGroupDigests>– XML-repesentations of DataGroupDigests elements
    <SignerInfos>
       <SignerInfo>
       ...
    </SignerInfos>       – XML-repesentations of SignerInfos elements
    <AA_KeyAlgorithm>    – text AA_KeyAlgorithm value
    <AA_Status>          – numeric AA_Status value
    <CA_Scheme>          – text CA_Scheme value
    <CA_KeyAlgorithm>    – text CA_KeyAlgorithm value
    <CA_Status>          – numeric CA_Status value
    <TA_Scheme>          – text TA_Scheme value
    <TA_Status>          – numeric TA_Status value
  </RFID_Authentication_Info>
```

6.3.52. TPassiveAuthenticationData

TRF_ TPassiveAuthenticationData structure is used to describe the parameters and the contents of DS-certificate required to verify the digital signature of EF.SOD security object when running in batch mode (see section 5.7.7).

```
struct TPassiveAuthenticationData
{
   TRF_FT_BYTES   Issuer;
   TRF_FT_BYTES   SerialNumber;
   TRF_FT_BYTES   SubjectKeyIdentifier;
   BYTE           *DS_Certificate;
   DWORD          DS_Certificate_Length;
};
```

Declaration: `RFID.h`

Fields:

`Issuer` – identifier of the organization that issued the certificate;

`SerialNumber` – certificate serial number;

`SubjectKeyIdentifier` – identifier of the signature subject;

`DS_Certificate` – binary array of the certificate contents;

`DS_Certificate_Length` – `DS_Certificate` length.

6.3.53. TRF_SOD_DG_Digest

TRF_SOD_DG_Digest structure is used to store the result of comparison of hash value computed by the data read from the RFID-chip with the value from EF.SOD for a single informational data group.

```
struct TRF_SOD_DG_Digest
{
   DWORD   DataGroup;
   DWORD   DigestCheckResult;
};
```

Declaration: RFID.h

Fields:

DataGroup – index of informational data group (value from 1 to 16)

DigestCheckResult – result of comparison for the given group (respective value from **eLDS_ParsingErrorCodes** or **eLDS_ParsingNotificationCodes**)

In XML-structures represented as a single node:

```
<DataGroupDigest DigestCheckResult="" DataGroup=""/>
```

Node attributes correspond to the fields of **TRF_SOD_DG_Digest** structure and contain their numerical values.

6.3.54. TRF_SOD_SignerInfo

TRF_SOD_SignerInfo structure is used to store the result of the verification of a single digital signature of the number present in EF.SOD.

```
struct TRF_SOD_SignerInfo
{
   DWORD                ErrorStatus;
   DWORD                nNotifications;
   DWORD                *Notifications;
   TRF_SOD_Certificate *Certificate;
   DWORD                version;
   TRF_FT_BYTES         DigestAlgorithm;
   TRF_FT_BYTES         SignatureAlgorithm;
};
```

Declaration: RFID.h

Field:

ErrorStatus – result of the digital signature verification (respective value from **eLDS_ParsingErrorCodes** or **eLDS_ParsingNotificationCodes**) nNotifications – number of **Notifications** array elements

Notifications – array of codes of non-critical annotations appeared during the process of the digital signature verification (respective values from **eLDS_ParsingErrorCodes** or **eLDS_ParsingNotificationCodes**)

Certificate – information on certificate corresponding to the verified digital signature

version – version of the digital signature data structure

DigestAlgorithm – digital signature hash-function algorithm identifier

SignatureAlgorithm – digital signature algorithm identifier

XML-representation of the structure:

```
<SignerInfo>
  <ErrorStatus>          – numeric ErrorStatus value
  <Notifications>
    <Notification Value=""/>
    ...
  </Notifications>       – numeric values of Notifications (in Value attributes)
  <Certificate>          – XML-representation of Certificate field
  <Version>              – numeric Version value
  <DigestAlgorithm>      – text DigestAlgorithm value
  <SignatureAlgorithm> – text SignatureAlgorithm value
</SignerInfo>
```

6.3.55. TRF_SOD_Certificate

TRF_SOD_Certificate structure is used to store information on a single certificate.

```
struct TRF_SOD_Certificate
{
  TRF_FT_BYTES  SignatureAlgorithm;
  TRF_FT_BYTES  Issuer;
  TRF_FT_BYTES  Subject;
  TRF_FT_BYTES  ValidFrom;
  TRF_FT_BYTES  ValidTo;
};
```

Declaration: RFID.h

Fields:

SignatureAlgorithm – digital signature algorithm identifier

Issuer – text information on the organization that issued the certificate, in the format

«<*code_of state*>, <*name_of organization*>, <*current_name*>»

Subject – text information on the organization that performed personalization of document, in the format

«<code_of state>, < name_of organization >, < current_name >»

`ValidFrom`	–	date of start of certificate validity, in «`YYYYMMDD`» format, where `YYYY` – year, `MM` – month, `DD` – day
`ValidTo`	–	certificate date of expiry, in «`YYYYMMDD`» format, where `YYYY` – year, `MM` – month, `DD` – day

XML-representation of the structure:

```
<Certificate>
  <SignatureAlgorithm> – text SignatureAlgorithm value
  <Issuer>             – text Issuer value
  <Subject>            – text Subject value
  <ValidFrom>          – text ValidFrom value
  <ValidTo>            – text ValidTo value
</Certificate>
```

6.3.56. TMIFARE_KeyTable

`TMIFARE_KeyTable` structure serves to transfer a set of authentication keys to the control library for use when reading data via MIFARE® Classic Protocol (see section 5.7.3).

```
struct TMIFARE_KeyTable
{
  BYTE KeyA[40][6];
  BYTE KeyB[40][6];
};
```

Declaration: `RFID.h`

Fields:

`KeyA`	–	key array `A`
`KeyB`	–	key array `B`

6.3.57. TRF_EID_TEXT_ARRAY

`TRF_EID_TEXT_ARRAY` structure is used to store the contents of informational data group of *`eID`* application as a text string [24, part 2, §A.2] (see section 5.8.10).

```
struct TRF_EID_TEXT_ARRAY
{
  BYTE          nType;
  TRF_FT_STRING Contents;
};
```

Declaration: `RFID.h`

Fields:

`nType` – ASN.1-tag of informational data group (**`eRFID_DataGroupTypeTag`**)

`Contents` – string contents (`UTF8` format is possible)

XML-representation of the structure:

```
<eID_DG1>
  <Type>
  <Contents>
</eID_DG1>
```

Name of the root node corresponds to the informational data group. Values and format of nodes correspond to the fields of **`TRF_EID_TEXT_ARRAY`** structure.

6.3.58. TRF_EID_GENERAL_PLACE

`TRF_EID_GENERAL_PLACE` structure is used to store the contents of informational data group of *`eID`* application represented as ASN.1 `GeneralPlace` object [24, part 2, §A.2] (see section 5.8.10). Describes the place of residence or birth.

```
struct TRF_EID_GENERAL_PLACE
{
  BYTE           nType;
  int            choice;
                                //choice==0
  TRF_FT_STRING  street;
  TRF_FT_STRING  city;
  TRF_FT_STRING  state;
  TRF_FT_STRING  country;
  TRF_FT_STRING  zipcode;
  TRF_FT_STRING  freetextPlace; //choice==1
  TRF_FT_STRING  noPlaceInfo;   //choice==2
};
```

Declaration: `RFID.h`

Fields:

`nType` – ASN.1-tag of informational data group (**`eRFID_DataGroupTypeTag`**)

`choice` – variant of the data group contents representation (values `0`, `1` or `2`)

Variant of representing the data with `choice = 0`:

`street` – street

`city` – city

`state` – region

`country` – country

`zipcode` – zip code

Variant of representing the data with `choice = 1`:

`freetextPlace` – text in free form with a description of some address (location)

Variant of representing the data with `choice = 2`:

`noPlaceInfo` – text in free form describing the reasons for the absence of information

For all strings `UTF8` format is allowed for the contents.

XML-representation of the structure:

```
<eID_DG9>
  <Type>
  <Street>
  <City>
  <State>
  <Country>
  <ZipCode>
  <FreetextPlace>
  <NoPlaceInfo>
</eID_DG9>
```

Name of the root node corresponds to the informational data group. Values and format of nodes correspond to the fields of **TRF_EID_GENERAL_PLACE** structure.

6.3.59. TRF_EID_TEXT

TRF_EID_TEXT structure is used to store the contents of informational data group of *eID* application represented as ASN.1 `Text` object [24, § E.2] (see section 5.8.10).

```
struct TRF_EID_TEXT
{
  BYTE            nType;
  int             choice;
                              //choice==0
  TRF_FT_STRING   uncompressed;
                              //choice==1
  TRF_FT_BYTES    compressed;
};
```

Declaration: `RFID.h`

Fields:

`nType` – ASN.1-tag of informational data group (**eRFID_DataGroupTypeTag**)

`choice` – variant of the data group contents representation (values `0` or `1`)

Variant of representing the data with `choice = 0`:

`uncompressed` – free `UTF8`-text.

Variant of representing the data with `choice = 1`:

`compressed` – free `UTF8`-text in compressed form (`DEFLATE` algorithm).

XML-representation of the structure:

```
<eID_DG19>
  <Type>
  <Uncompressed>
  <Compressed>
</eID_DG19>
```

Name of the root node corresponds to the informational data group. Values and format of nodes correspond to the fields of **`TRF_EID_TEXT`** structure.

6.3.60. TRF_EID_OPTIONAL_DATA

`TRF_EID_OPTIONAL_DATA` structure is used to store the contents of informational data group of *`eID`* application represented as a list of ASN.1 `OptionalData` objects [24, part 2, §A.2] (see section 5.8.10).

```
struct TRF_EID_OPTIONAL_DATA
{
  BYTE                          nType;
  BYTE                          nRecords;
  TRF_EID_OPTIONAL_DATA_ITEM    **pDataArray;
};
```

Declaration: `RFID.h`

Fields:

`nType` – ASN.1-tag of informational data group (**`eRFID_DataGroupTypeTag`**)

`nRecords` – number of `pDataArray` array elements

`pDataArray` – array of optional data elements

XML-representation of the structure:

```
<eID_DG12>
  <Type>
  <DataArray>
     <OptionalDataItem>
     ...
  </DataArray>
</eID_DG12>
```

Name of the root node corresponds to the informational data group. Values and format of nodes correspond to the fields of **`TRF_EID_OPTIONAL_DATA`** structure. The number of `DataArray` elements corresponds to the number of elements in `pDataArray` array.

6.3.61. TRF_EID_OPTIONAL_DATA_ITEM

TRF_EID_OPTIONAL_DATA_ITEM structure is used to store the contents of a single element of the list of optional data of *eID* application information group.

```
struct TRF_EID_OPTIONAL_DATA_ITEM
{
   TRF_FT_STRING type_id;
   TRF_FT_BYTES  data;
};
```

Declaration: `RFID.h`

Fields:

`type_id`	–	identifier of the optional data type (OID)
`data`	–	binary array of the optional data element contents

XML-representation of the structure:

```
<OptionalDataItem>
  <TypeID>
  <Data Length="">
      <![CDATA[]]>
  </Data>
</OptionalDataItem>
```

Values and format of nodes correspond to the fields of **TRF_EID_OPTIONAL_DATA_ITEM** structure.

6.3.62. TRFID_AntennaParamsPair

TRFID_AntennaParamsPair structure is used to describe a single element of the list of RFID-chip reader antenna parameters (see section 5.4.5).

```
union TRFID_AntennaParamsPair
{
   BYTE RawParams[8];
   struct
   {
      TRFID_AntennaParams  Layer3;
      TRFID_AntennaParams  Layer4;
   };
};
```

Declaration: `RFID.h`

Fields:

`RawParams`	–	representation of the parameters in the form of one-dimensional array of bytes
`Layer3`	–	set of antenna parameters for the commands of `Layer 3` level

`Layer4` – set of antenna parameters for the commands of `Layer 4` level

6.3.63. TRFID_AntennaParams

`TRFID_AntennaParamsPair` structure is used to describe a set of antenna parameters of the RFID-chip reader for the commands of a single level (see section 5.4.5).

```
union TRFID_AntennaParams
{
   BYTE RawParams[4];
   struct{
      BYTE SensA;
      BYTE KRecA;
      BYTE SensB;
      BYTE KRecB;
   };
};
```

Declaration: `RFID.h`

Fields:

`RawParams` – representation of the parameters in the form of one-dimensional array of bytes

`SensA` – coefficient of sensitivity when working with chips of type `A`

`KRecA` – gain when working with chips of type `A`

`SensB` – coefficient of sensitivity when working with chips of type `B`

`KRecB` – gain when working with chips of type `B`

6.3.64. TCustomRawDataList

`TCustomRawDataList` structure is used to describe a list of containers for storing binary data arrays.

```
struct TCustomRawDataList
{
   int                       Count;
   TCustomRawDataToParse    *pArray;
};
```

Declaration: `RFID.h`

Fields:

`Count` – number of `pArray` array elements

`pArray` – number of array objects

6.3.65. TCustomRawData / TCustomRawDataToParse

`TCustomRawData` structure is used to describe a binary data array.

```
typedef struct TCustomRawData TCustomRawDataToParse;

struct TCustomRawData
{
  BYTE   *buffer;
  DWORD  length;
};
```

Declaration: `RFID.h`

Fields:

`buffer` – binary data array

`length` – length of `buffer` array

6.3.66. TRFID_Session

`TRFID_Session` structure is used to describe the results of work with the SDK within the context of the current communication session with electronic document (see section 5.8).

```
struct TRFID_Session
{
  DWORD                        VirtualMode;
  DWORD                        SDKVersion;
  DWORD                        DriverVersion;
  DWORD                        FirmwareVersion;
  DWORD                        RFControlMode;
  TRFID_CardPropertiesExt CardProperties;
  TRFID_AntennaParamsPair AntennaSetup;
  DWORD                        ExtLeSupport;
  DWORD                        TotalBytesSent;
  DWORD                        TotalBytesReceived;
  DWORD                        ProcessTime;
  TRFID_AccessKey              Session_key;
  TRFID_Terminal               Session_terminal;
  DWORD                        Session_procedure;
  TRFID_AccessKey              Session_eSignPIN;
  TTerminalVerificationData VerifiedData;
  TRFID_Items_List             *pRootFiles;
  TRFID_Items_List             *pApplications;
  int                          ActiveApplicationIdx;
  TRFID_Items_List             *pAccessControls;
  TRFID_Items_List             *pSecurityObjects;
  DWORD                        Status;
};
```

Declaration: `RFID.h`

Field:

`VirtualMode` – sign of virtual session when working with loaded data from a previous communication session with the electronic document (see section 5.8.22)

`SDKVersion`	–	version of the main control library (see section 6.1.7)
`DriverVersion`	–	version of RFID-reader driver (see section 5.3) in 'A.B.C.D' format, where • A = HIBYTE(HIWORD()) • B = LOBYTE(HIWORD()) • C = HIBYTE(LOWORD()) • D = LOBYTE(LOWORD())
`FirmwareVersion`	–	RFID-reader firmware version (see section 5.3) in 'A.B' format, where • A = HIBYTE(LOWORD()) • B = LOBYTE(LOWORD())
`RFControlMode`	–	working mode of the main control library (see sections 5.4.1, 5.4.2). Combination of **eRFID_ControlRF** values
`CardProperties`	–	set of electronic document chip characteristics (see section 5.8.3)
`AntennaSetup`	–	not used since SDK version 3.5;
`ExtLeSupport`	–	sign of support of RFID-chip for extended length commands of reading (see section 5.4.4) (`RFID_Error_NotPerformed`, `RFID_Error_NotAvailable` or `RFID_Error_NoError`)
`TotalBytesSent`	–	total number of bytes transmitted to the RFID-chip during the whole session
`TotalBytesReceived`	–	total number of bytes received from the RFID-chip during the whole session
`ProcessTime`	–	total time of all operations performed during the session, ms
`Session_key`	–	used secure data access key (see section 5.8.6)
`Session_terminal`	–	terminal configuration (see section 5.8.4)
`Session_procedure`	–	type of performed document authentication procedure (see section 5.8.5)
`Session_eSignPIN`	–	used access key to the functions of *eSign* application (see section 5.8.21)
`VerifiedData`	–	contents of the verified auxiliary data (see section 5.8.18)
`pRootFiles`	–	list of containers to store information about the read files of the root *Master File* (see section 5.8.10). List elements are **TRFID_DataFile ***
`pApplications`	–	list of containers to store information about the involved applications of electronic document (see section 5.8.9). List elements are **TRFID_Application ***
`ActiveApplicationIdx`	–	index of the current active application (see section 5.8.9)
`pAccessControls`	–	list of containers to store information about the supported procedures of authentication and secure data access within the context of the session (see sections 5.8.3, 5.8.7). List elements are **TRFID_AccessControlInfo ***
`pSecurityObjects`	–	list of containers to store information about the detected document security objects (see section 5.8.12). List elements are **TRFID_SecurityObject ***

`Status` – result of the last session operation (see section 5.8.2). One of **eRFID_ErrorCodes** values, coinsides with the return code from the last **_ExecuteCommand()** call

XML-representation of the structure:

```
<RFID_Session_Data>
  <VirtualMode>        – boolean VirtualMode value
  <SDKVersion>         – text SDKVersion value in format 'A.B' (e.g. "3.1")
  <DriverVersion>      – text DriverVersion value in format 'A.B.C.D'
                         (e.g. "6.2.5.4")
  <FirmwareVersion>    – text FirmwareVersion value in format 'A.B'
                         (e.g. "5.19")
  <RFControlMode>      – numeric RFControlMode value in hexadecimal format
                         (e.g. "0x00000040")
  <CardProperties>     – XML-representation of CardProperties field
  <AntennaSetup>       – not used since SDK version 3.5
  <ExtLeSupport>       – text representation of ExtLeSupport field in the format
                         "S1 [S2]", where S1 – code abbreviation,
                         S2 – numeric value in hexadecimal format
                         (e.g. "RFID_Error_NoError [0x00000001]")
  <TotalBytesSent>     – numeric TotalBytesSent value
  <TotalBytesReceived> – numeric TotalBytesReceived value
  <ProcessTime>        – numeric ProcessTime value
  <Session_key>        – XML-representation of Session_key field
  <Session_terminal>   – XML-representation of Session_terminal field
  <Session_procedure>  – text abbreviation of the value from Ses-
                         sion_procedure field
  <Session_eSignPIN>   – XML-representation of Session_eSignPIN field
  <VerifiedData>       – XML-representation of VerifiedData field
  <RootFiles>
     <RFID_DataFile>
     ...
  </RootFiles>         – pRootFiles list contents
  <Applications>
     <RFID_Application>
     ...
  </Applications>      – pApplications list contents
  <AccessControls>
     <RFID_AccessControlInfo>
     ...
  </AccessControls>    – pAccessControls list contents
  <SecurityObjects>
     <RFID_SecurityObject>
     ...
  </SecurityObjects>   – pSecurityObjects list contents
```

`<Status>` – text representation of `Status` field in the format `"S1 [S2]"`, where S1 –code abbreviation, S2 – numeric value in hexadecimal format (e.g. `"RFID_Error_NoError [0x00000001]"`)

`</RFID_Session_Data>`

6.3.67. TRFID_Application

`TRFID_Application` structure is used to describe the contents of a single LDS application and their analysis within the context of the communication session with electronic document (see section 5.8).

```
struct TRFID_Application
{
   DWORD             nType;
   DWORD             Status;
   TRF_FT_BYTES      ApplicationID;
   TRF_FT_BYTES      Version;
   TRF_FT_BYTES      UnicodeVersion;
   TRF_FT_BYTES      DataHashAlgorithm;
   TRFID_Items_List *pFiles;
};
```

Declaration: `RFID.h`

Fields:

`nType` – type of application of electronic document (one of **`eRFID_Application_Type`** values)

`Status` – status of the application selection procedure (see section 5.8.9) (one of **`eRFID_ErrorCodes`** values)

`ApplicationID` – application identifier

`Version` – application version

`UnicodeVersion` – Unicode version for application

`DataHashAlgorithm` – algorithm for calculating hash values for files for the procedure of PA (see section 5.8.13)

`pFiles` – list of containers to store information about the read files of the application (see section 5.8.10). List elements are **`TRFID_DataFile *`**

For *`ePassport`* application the information on the application version and the Unicode version is received during analysis of `EF.COM` data [2, section III, § 14], for *`eID`* application – `EF.CardSecurity/EF.ChipSecurity` data, containing the corresponding `eIDSecurityInfo` ASN.1-object [24, part 3, §A.1.1.6].

XML-representation of the structure:

```
<RFID_Application>
  <Type>                 – text abbreviation of the value from Type field
  <ApplicationID>        – XML-representation of ApplicationID field in text
                           format. Each byte of ApplicationID represented by
                           its hexadecimal value. The individual bytes are separated
                           by spaces (e.g. "A0 00 00 02 47 10 01")
  <Status>               – text representation of Status field in the format "S1
                           [S2]", where S1 – code abbreviation, S2 – numeric val-
                           ue in hexadecimal format (e.g.
                           "RFID_Error_NoError [0x00000001]")
  <Version>              – text Version value (e.g. "0107")
  <UnicodeVersion>       – text UnicodeVersion value (e.g. "040000")
  <DataHashAlgorithm>    – text DataHashAlgorithm value in the format "S1
                           (S2)", where S1 – algorithm name, S2 – algorithm
                           identifier (OID string).
  <Files>
     <RFID_DataFile>
     ...
  </Files>               – pRootFiles list contents
</RFID_Application>
```

6.3.68. TRFID_DataFile

TRFID_DataFile structure is used to describe the contents of a single file of the LDS of electronic document and the analysis of its contents within the context of the communication session with electronic document (see section 5.8).

```
struct TRFID_DataFile
{
  TRF_FT_BYTES              FileID;
  DWORD                     nType;
  TRFID_Application         *pApplication;
  TRF_FT_BYTES              FileData;
  void                      *pParsedData;
  TRFID_Items_List          *pParsingNotifications;
  DWORD                     ReadingStatus;
  DWORD                     nReadingTime;
  TRFID_Items_List          *pNotifications;
  DWORD                     PA_Status;
  TDocVisualExtendedInfo    *pDocFields_Text;
  TDocGraphicsInfo          *pDocFields_Graphics;
  TOriginalRFIDGraphicsInfo *pDocFields_Originals;
};
```

Declaration: `RFID.h`

Fields:

`FileID`	–	file identifier
`nType`	–	type of the file (of the object) of data (one of **`eRFID_DataFile_Type`** values)
`pApplication`	–	reference to the object of the parent application. Contains an element of `TRFID_Session.pApplications`
`FileData`	–	binary array of the file contents
`pParsedData`	–	pointer to the structure of logically parsed data (see section 5.8.10)
`pParsingNotifications`	–	list of remarks arisen when making logical analysis of the data contents. List elements are `DWORD` values, corresponding to the constants from **`eLDS_ParsingNotificationCodes`**
`ReadingStatus`	–	status of the physical file reading (one of **`eRFID_ErrorCodes`** values)
`nReadingTime`	–	file reading time, ms
`pNotifications`	–	list of remarks arisen when reading data from the memory of the chip and analysing their ASN.1-structure. List elements are `DWORD` values, corresponding to the constants from the **`eLDS_ParsingErrorCodes`**
`PA_Status`	–	result of the data integrity verification within the context of PA (see section 5.8.13)
`pDocFields_Text`	–	list of document text fields formed on the basis of the file contents (see sections 5.8.10, 5.9)
`pDocFields_Graphics`	–	list of document graphic fields formed on the basis of the file contents (see sections 5.8.10, 5.9)
`pDocFields_Originals`	–	list of the original binary representation of graphic document fields formed on the basis of the file contents (see sections 5.8.10, 5.9)

XML-representation of the structure:

<RFID_DataFile>

`<Type>`	–	text abbreviation of the value from `Type` field
`<FileData>`	–	XML-representation of `FileData` field
`<ReadingStatus>`	–	text representation of `ReadingStatus` field
`<FileID>`	–	XML-representation of `FileID` field in text format. Each byte of `FileID` represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"01 1E"`)
`<ReadingTime>`	–	numeric `ReadingTime` value

```
<ParsedData>
   <ParsingNotifications>
       <Item>
       ...
   </ParsingNotifications>
</ParsedData>
```
– `pParsingNotifications` list contents

```
<Notifications>
   <Item>
```

```
      ...
    </Notifications>    – pNotifications list contents
    <PA_Status>         – text representation of PA_Status field
    <DocFields_Text>
      <Field>
      ...
    </DocFields_Text>   – the list of abbreviations of types of the fields registered
                          in pDocFields_Text
    <DocFields_Graphics>
      <Field>
      ...
    </DocFields_Graphics>  – the list of abbreviations of types of the fields regis-
                          tered in pDocFields_Graphics
    <DocFields_Originals>
      <Field>
      ...
    </DocFields_Originals> – the list of abbreviations of types of the fields regis-
                          tered in pDocFields_Originals
  </RFID_DataFile>
```

ReadingStatus and PA_Status elements are strings in the format "S1 [S2]", where S1 – abbreviation of the corresponding status code, S2 – the numeric value in hexadecimal format (e.g. "RFID_Error_NoError [0x00000001]").

Item elements of ParsingNotifications and Notifications lists are strings in the format "S1 [S2]", where S1 – abbreviation of the corresponding notification code, S2 – the numeric value in hexadecimal format (e.g. "ntfLDS_ICAO_Certificate_Validity [0x9000020C]").

6.3.69. TRFID_AccessControlInfo

TRFID_AccessControlInfo structure is used to describe the results of a single authentication procedure or a procedure of secure data access within the context of the communication session with electronic document (see sections 5.8).

```
struct TRFID_AccessControlInfo
{
  DWORD             dwType;
  DWORD             Status;
  TRFID_Items_List  *pOptions;
  int               ActiveOptionIdx;
  TRFID_Items_List  *pNotifications;
  void              *pProcedure;
  TRF_FT_BYTES      SpecificData1;
  TRF_FT_BYTES      SpecificData2;
};
```

Declaration: `RFID.h`

Field:

`dwType` – procedure type (one of **`eRFID_AccessControl_ProcedureType`** values)

`Status` – procedure status (`RFID_Error_NotPerformed`, `RFID_Error_NotAvailable`, `RFID_Error_NoError` or the error code from **`eRFID_ErrorCodes`**)

`pOptions` – list of containers to store information about the available variants of the procedure (see sections 5.8.3, 5.8.7). List elements are **`TRFID_AccessControl_Option *`**

`ActiveOptionIdx` – index of the active variant of the procedure (see section 5.8.7)

`pNotifications` – list of remarks arisen during the procedure. The elements of the list are `DWORD` values, corresponding to the constants from **`eLDS_ParsingErrorCodes`**

`pProcedure` – for internal SDK use

`SpecificData1` – container for storage of procedure specific data

`SpecificData2` – container for storage of procedure specific data

When performing TA in step-by-step mode on the second step in `SpecificData1` field of the procedure description object the contents of *challenge* is stored, `SpecificData2` contains the contents of its hash value (see section 5.8.15).

`SpecificData1` and `SpecificData2` fields in the RI procedure description object are used to store the received terminal sector identifiers (see section 5.8.17).

XML-representation of the structure:

```
<RFID_AccessControlInfo>
  <Type>                – text abbreviation of the value from dwType field
  <Status>              – text representation of Status field
  <Notifications>
     <Item>
     ...
  </Notifications>      – pNotifications list contents
  <AccessControlOptions>
     <RFID_AccessControl_Option>
     ...
  </AccessControlOptions>  –  pOptions list contents
  <ActiveOptionIdx>     – numeric ActiveOptionIdx value
  <SectorID1>           – XML-representation of SpecificData1 field (for RI
                          procedure)
  <SectorID2>           – XML-representation of SpecificData2 field (for RI
                          procedure)
  <Challenge>           – XML-representation of SpecificData1 field (for TA
                          procedure)
```

```
    <HashValue>          – XML-representation of SpecificData2 field (for TA
                           procedure)
  </RFID_AccessControlInfo>
```

`Status` element is a string in the format "`S1 [S2]`", where S1 – abbreviation of the status code, S2 – the numeric value in hexadecimal format (e.g. "`RFID_Error_NoError [0x00000001]`").

`Item` elements of `Notifications` list are strings in the format "`S1 [S2]`", where S1 – abbreviation of the corresponding notification code, S2 – the numeric value in hexadecimal format (e.g. "`ntfLDS_CVCertificate_Validity [0x91000202]`").

6.3.70. TRFID_AccessControl_Option

`TRFID_AccessControl_Option` structure is used to describe a single variant of authentication or secure data access procedure performance within the context of the communication session with electronic document (see section 5.8).

```
struct TRFID_AccessControl_Option
{
   DWORD            Version;
   TRF_FT_BYTES     Scheme;
   TRF_FT_BYTES     KeyAlgorithm;
   DWORD            ChipIndividual;
};
```

Declaration: `RFID.h`

Fields:

`Version`	–	procedure version (for PACE, CA, TA)
`Scheme`	–	algorithm of used cryptographic scheme (CA, TA, AA), TA public key algorithm (is specified after the procedure itself), identifier if standardized public key domain parameters (for PACE), URL (for Card Info)
`KeyAlgorithm`	–	public key algorithm (for PACE, CA, AA, RI), identifier of the working CVCA-key (for TA), FID (for Card Info)
`ChipIndividual`	–	sign of the accessibility of key usage for privileged terminals only (for CA) [24, §3.2.4, §A.1.2]. For informational Card Info procedure low -order byte contains SFID.

XML-representation of the structure:

- for PACE (`acptPACE`) procedure

```
  <RFID_AccessControl_Option>
    <Version>            – numeric Version value
    <StdDomainParams>    – text Scheme value
```

```
  <KeyAlgorithm>          – text KeyAlgorithm value
</RFID_AccessControl_Option>
```

StdDomainParams value is in the form "S1 [S2]", where S1 - the name of a standardized set of parameters, S2 - set identifier in hexadecimal.

KeyAlgorithm value is in the form "S1 (S2)", where S1 – algorithm name, S2 – algorithm identifier (OID string).

- for CA (acptCA), AA (acptAA) and RI (acptRI) procedures

```
<RFID_AccessControl_Option>
  <Version>               – numeric Version value
  <Scheme>                – text Scheme value
  <KeyAlgorithm>          – text KeyAlgorithm value
  <ChipIndividual>        – boolean ChipIndividual value
</RFID_AccessControl_Option>
```

Scheme and KeyAlgorithm values are in the form "S1 (S2)", where S1 – algorithm name, S2 – algorithm identifier (OID string).

- for TA (acptTA) procedure

```
<RFID_AccessControl_Option>
  <Version>               – numeric Version value
  <Scheme>                – text Scheme value
  <CAR>                   – text KeyAlgorithm value
</RFID_AccessControl_Option>
```

- for Card Info (acptCardInfo) informational procedure (see section 5.8.3)

```
<RFID_AccessControl_Option>
  <URL>                   – text Scheme value
  <FID>                   – text KeyAlgorithm value
  <SFID>                  – hexadecimal value of the low-order byte of
                            ChipIndividual field
</RFID_AccessControl_Option>
```

6.3.71. TRFID_SecurityObject

TRFID_SecurityObject structure is used to describe the contents of a single document security object (SO) and the results of its check within the context of the communication session with electronic document (see section 5.8).

```
struct TRFID_SecurityObject
{
```

```
  DWORD              Version;
  TRF_FT_BYTES       ObjectType;
  TRFID_DataFile    *pFileReference;
  TRFID_Items_List  *pSignerInfos;
  TRFID_Items_List  *pNotifications;
};
```

Declaration: `RFID.h`

Field:

`Version` – security object version (version of `LDSSecurityObject` ASN.1-object for *ePassport* application [2, §A3.1])

`ObjectType` – security object identifier (OID from `LDSSecurityObject` object for *ePassport* application [2, §A3.1], `szOID_BSI_SecurityObject` for *eID* application)

`pFileReference` – reference to the source file of the security object data

`pSignerInfos` – list of containers to store information about digital signature objects contained in the SO (see section 5.8.12). The elements of the list are **TRFID_SignerInfo ***

`pNotifications` – list of remarks arisen during the analysis of SO data structure. The elements of the list are `DWORD` values, corresponding to the constants from **eLDS_ParsingErrorCodes**

XML-representation of the structure:

```
<RFID_SecurityObject>
  <Version>           – numeric Version value
  <ObjectType>        – text value of ObjectType field in format "S1 (S2)",
                        where S1 – security object name, S2 – identifier (OID
                        string)
  <FileReference>     – text abbreviation of the file type from pFileReference
                        field
  <SignerInfos>
     <RFID_SignerInfo_Ex>
     ...
  </SignerInfos>      – pSignerInfos list contents
  <Notifications>
     <Item>
     ...
  </Notifications>    – pNotifications list contents
</RFID_SecurityObject>
```

`Item` elements of `Notifications` list are strings in the format "`S1 [S2]`", where S1 – abbreviation of the corresponding notification code, S2 – the numeric value in a hexadecimal format.

6.3.72. TRFID_SignerInfo_Ex

TRFID_SignerInfo_Ex structure is used to describe the contents of a single copy of digital signature of the document security object and the results of its check within the context of the communication session with electronic document (see section 5.8.12). Corresponds to SignerInfo ASN.1-object [7, §5.3].

```
struct TRFID_SignerInfo_Ex
{
   DWORD                        Version;
   TRFID_DistinguishedName *pIssuer;
   TRF_FT_BYTES                 SerialNumber;
   TRF_FT_BYTES                 SubjectKeyIdentifier;
   TRF_FT_STRING                DigestAlgorithm;
   TRFID_Items_List            *pSignedAttributes;
   TRF_FT_STRING                SignatureAlgorithm;
   TRF_FT_BYTES                 Signature;
   TRFID_Items_List            *pUnsignedAttributes;
   DWORD                        PA_Status;
   TRFID_Items_List            *pCertificateChain;
   TRF_FT_BYTES                 DataToHash;
   TRFID_Items_List            *pNotifications;
};
```

Declaration: RFID.h

Fields:

Version	–	version of SignerInfo ASN.1 structure;
pIssuer	–	identifier of the necessary certificate issuer;
SerialNumber	–	serial number of the necessary certificate;
SubjectKeyIdentifier	–	signature object identifier of the necessary certificate;
DigestAlgorithm	–	hash algorithm identifier (OID) for digital signature generation;
pSignedAttributes	–	list of the signed attributes. Elements of the list are **TRFID_Attribute_Data ***;
SignatureAlgorithm	–	digital signature algorithm identifier (OID);
Signature	–	binary data of the verified digital signature;
pUnsignedAttributes	–	list of the unsigned attributes. Elements of the list are **TRFID_Attribute_Data ***;
PA_Status	–	result of the digital signature verification (RFID_Error_NotPerformed, RFID_Error_NoError либо RFID_Error_Failed);
pCertificateChain	–	certificate chain, used for the digital signature verification. Elements of the list are **TRFID_Certificate_Ex ***;
DataToHash	–	binary data array used to calculate the hash value for digital signature verification;

`pNotifications` – list of remarks arisen during the analysis of the data structure and performance of digital signature verification. The elements of the list are `DWORD` values, corresponding to the constants from **`eLDS_ParsingErrorCodes`**.

XML-representation of the structure:

```
<RFID_SignerInfo_Ex>
  <Version>
  <Issuer>
  <SerialNumber>
  <SubjectKeyIdentifier>
  <DigestAlgorithm>
  <SignedAttributes>
     < RFID_Attribute_Data>
     ...
  </SignedAttributes>
  <SignatureAlgorithm>
  <Signature>
  <PA_Status>
  <CertificateChain>
     <RFID_Certificate_Ex>
     ...
  </CertificateChain>
  <DataToHash>
  <Notifications>
     <Item>
     ...
  </Notifications>
</RFID_SignerInfo_Ex>
```

Values and format of nodes correspond to the fields of **`TRFID_SignerInfo_Ex`**. The number of `SignedAttributes` elements corresponds to the number of elements in `pSignedAttributes` array, the number of `CertificateChain` elements – to `pCertificateChain` array.

`DigestAlgorithm` and `SignatureAlgorithm` elements are strings in the format "`S1 (S2)`", where S1 – algorithm name, S2 – identifier (OID string).

`PA_Status` is a string in the format "`S1 [S2]`", where S1 – status code abbreviation, S2 – numeric value in a hexadecimal format.

`Item` elements of `Notifications` list are strings in the format "`S1 [S2]`", where S1 – abbreviation of the corresponding notification code, S2 – the numeric value in a hexadecimal format.

6.3.73. TRFID_Certificate_Ex

`TRFID_Certificate_Ex` structure is used to describe the certificate contents used for the digital signature verification of the document security object within the context of the communication session with electronic document (see section 5.8.12). Corresponds to `Certificate` ASN.1-object [6, §4].

```
struct TRFID_Certificate_Ex
{
   DWORD                        Version;
   TRF_FT_BYTES                 SerialNumber;
   TRF_FT_STRING                SignatureAlgorithm;
   TRFID_DistinguishedName     *pIssuer;
   TRFID_Validity              *pValidity;
   TRFID_DistinguishedName     *pSubject;
   TRF_FT_STRING                SubjectPKAlgorithm;
   TRFID_Items_List            *pExtensions;
   TRFID_Items_List            *pNotifications;
   DWORD                        Origin;
   DWORD                        Type;
   TRF_FT_STRING                FileName;
   DWORD                        PA_Status;
   TRFID_RevocationInfo        *pRevocationInfo;
   TRFID_Certificate_Ex        *pIssuerCertificate;
};
```

Declaration: `RFID.h`

Fields:

`Version` – version of `Certificate` ASN.1 structure;

`SerialNumber` – certificate serial number;

`SignatureAlgorithm` – certificate digital signature algorithm identifier (OID);

`pIssuer` – identifier of the certificate issuer;

`pValidity` – certificate validity period;

`pSubject` – identifier of the signature subject;

`SubjectPKAlgorithm` – certificate public key algorithm identifier (OID);

`pExtensions` – list of the certificate extensions. Elements of the list are **`TRFID_PKI_Extension *`**;

`pNotifications` – list of remarks arisen during the analysis of the certificate data structure and its validity verification. The elements of the list are `DWORD` values, corresponding to the constants from **`eLDS_ParsingErrorCodes`**;

`Origin` – certificate origin (one of **`eRFID_CertificateOrigin`** values);

`Type` – certificate type (one of **`eRFID_CertificateType`** values);

`FileName` – the name of the certificate source file, if there is one (UTF8 string);

`PA_Status` – result of certificate's digital signature verification (`RFID_Error_NotPerformed`, `RFID_Error_NoError` либо `RFID_Error_Session_PA_SignatureCheckFailed`);

`pRevocationInfo` – reference to the object with certificate revocation information. If not revoked, contains 0;

`pIssuerCertificate` – reference to the parent certificate. Possible reference to itself in the case of a self-signed certificate.

XML-representation of the structure:

```
<RFID_Certificate_Ex>
  <Origin>
  <Type>
  <FileName>
  <PA_Status>
  <Version>
  <SerialNumber>
  <SignatureAlgorithm>
  <Issuer>
  <Validity>
  <Subject>
  <SubjectPKAlgorithm>
  <Extensions>
     <RFID_PKI_Extension>
     ...
  </Extensions>
  <Notifications>
     <Item>
     ...
  </Notifications>
</RFID_Certificate_Ex>
```

Values and format of nodes correspond to the fields of **`TRFID_Certificate_Ex`**. The number of `Extensions` elements corresponds to the number of elements in `pExtensions` array.

`Origin` and `Type` elements contain text abbreviations of the values of the corresponding fileds.

`SignatureAlgorithm` and `SubjectPKAlgorithm` values are strings in the format "`S1 (S2)`", where S1 – algorithm name, S2 – identifier (OID string).

`PA_Status` is a string in the format "`S1 [S2]`", where S1 – status code abbreviation, S2 – numeric value in a hexadecimal format.

`Item` elements of `Notifications` list are strings in the format "`S1 [S2]`", where S1 – abbreviation of the corresponding notification code, S2 – the numeric value in a hexadecimal format.

6.3.74. TRFID_Items_List

TRFID_Items_List is used to describe the list of any elements.

```
struct TRFID_Items_List
{
  DWORD   dwCount;
  void    **pItems;
};
```

Declaration: `RFID.h`

Fields:

`dwCount`	–	number of `pItems` array elements
`pItems`	–	array of pointers to the structures

The specific type of list elements is defined by the context of the object.

6.3.75. TRFID_DistinguishedName

TRFID_DistinguishedName structure contains information that serves as the distinguished name (identifier) of an object. Corresponds to `Name` ASN.1-object [X.501][6, §4.1.2.4].

```
struct TRFID_DistinguishedName
{
  TRF_FT_BYTES      Data;
  TRF_FT_STRING     FriendlyName;
  TRFID_Items_List *pAttributes;
};
```

Declaration: `RFID.h`

Fields:

`Data`	–	contents of the identifier in binary form;
`FriendlyName`	–	text representation of the identifier (UTF8);
`pAttributes`	–	list of individual attributes contained in the identifier. The elements of the list are **TRFID_Attribute_Name ***.

In XML-structures appears as a separate node, named based on the context of use, and contains the following elements:

```
<node name>
  <Data>
  <FriendlyName>
  <Attributes>
     <RFID_Attribute_Name>
     ...
  </Extensions>
</node name>
```

Values and format of nodes correspond to the fields of **TRFID_DistinguishedName** structure. The number of Attributes elements corresponds to the number of elements in pAttributes array.

6.3.76. TRFID_Attribute_Name

TRFID_Attribute_Name contains the data of one attribute which is part of the distinguished name. Corresponds to AttributeTypeAndValue ASN.1-object [X.501][6, §4.1.2.4].

```
struct TRFID_Attribute_Name
{
  TRF_FT_STRING     Type;
  TRF_FT_STRING     Value;
};
```

Declaration: RFID.h

Fields:

Type – attribute identifier (OID ASCII string);

Value – text value of the attribute (UTF8).

XML-representation of the structure:

<RFID_Attribute_Name>

<Type> – contents of the identifier in the format "S1 (S2)", where S1 – attribute name, S2 – identifier (OID string);

<Value> – text value of Value field

</RFID_Attribute_Name>

6.3.77. TRFID_Attribute_Data

TRFID_Attribute_Data structure contains the data of one attribute of the digital signature object. Corresponds to Attribute ASN.1-object [7, §5.3].

```
struct TRFID_Attribute_Data
{
  TRF_FT_STRING     Type;
  TRF_FT_BYTES      Data;
  void             *pParsedData;
};
```

Declaration: RFID.h

Fields:

Type – attribute identifier (OID ASCII string);

Data – attribute binary data;

pParsedData – reserved.

XML-representation of the structure:

```
<RFID_Attribute_Data>
  <Type>      – contents of the identifier in the format "S1 (S2)", where S1 – at-
                tribute name, S2 – identifier (OID string);
  <Data>      – contents of Data field
</RFID_Attribute_Data>
```

6.3.78. TRFID_Validity

TRFID_Validity structure contains information on a certificate validity. Corresponds to Validity ASN.1-object [6, §4.1].

```
struct TRFID_Validity
{
  TRF_FT_STRING     NotBefore;
  TRF_FT_STRING     NotAfter;
};
```

Declaration: RFID.h

Fields:

NotBefore – string of the start date;

NotAfter – string of the expiration date.

The format of the strings is defined by [6, §4.1.2.5] and can be YYMMDDHHMMSSZ (in the case of using UTCTime format) or YYYYMMDDHHMMSSZ (in the case of GeneralizedTime format).

In XML-structures appears as a separate node, named based on the context of use, and contains the following elements:

```
<node name>
  <NotBefore>
  <NotAfter>
</node name>
```

Values and format of nodes correspond to the fields of **TRFID_Validity** structure.

6.3.79. TRFID_PKI_Extension

TRFID_PKI_Extension structure contains the data of a certificate extension. Corresponds to Extension ASN.1-object [6, §4.1].

```
struct TRFID_PKI_Extension
{
  TRF_FT_STRING     Type;
  TRF_FT_BYTES      Data;
  void             *pParsedData;
};
```

Declaration: `RFID.h`

Fields:

`Type` – extension identifier (OID ASCII string);

`Data` – extension binary data;

`pParsedData` – reserved.

XML-representation of the structure:

```
<RFID_PKI_Extension>
  <Type>      – contents of the identifier in the format "S1 (S2)", where S1 – attribute name, S2 – identifier (OID string);
  <Data>      – contents of Data field
</RFID_PKI_Extension>
```

6.3.80. TRFID_RevocationInfo

`TRFID_RevocationInfo` structure contains the information on the certificate revocation. Corresponds to the element of the list of revoked certificates `TBSCertList` ASN.1-object [6, §5.1].

```
struct TRFID_RevocationInfo
{
  TRFID_CRL_Ex      *pOwner;
  TRF_FT_BYTES       Certificate;
  TRF_FT_STRING      RevocationDate;
  TRFID_Items_List *pEntryExtensions;
};
```

Declaration: `RFID.h`

Fields:

`pOwner` – reference to the parent Certificate Revocation List (CRL) object;

`Certificate` – revoked certificate serial number;

`RevocationDate` – certificate revocation date;

`pEntryExtensions` – extensions list of CRL. Elements of the list are **`TRFID_PKI_Extension *`**.

The format of the string `RevocationDate` is defined by [6, §4.1.2.5] and can be `YYMMDDHHMMSSZ` (in the case of using `UTCTime` format) or `YYYYMMDDHHMMSSZ` (in the case of `GeneralizedTime` format).

XML-representation of the structure:

```
<RFID_RevocationInfo>
  <CertificateSerialNumber>
  <RevocationDate>
  <EntryExtensions>
     <RFID_PKI_Extension>
     ...
```

```
    </EntryExtensions>
    <Owner_CRL>
  </RFID_RevocationInfo>
```

Values and format of nodes correspond to the fields of **TRFID_RevocationInfo** structure. The number of Extensions elements corresponds to the number of elements in pExtensions array.

6.3.81. TRFID_CRL_Ex

TRFID_CRL_Ex structure is used to describe the contents of the certificate revocation list (CRL). Corresponds to TBSCertList ASN.1-object [6, §5].

```
struct TRFID_CRL_Ex
{
  DWORD                           Version;
  TRF_FT_STRING                   SignatureAlgorithm;
  TRFID_DistinguishedName        *pIssuer;
  TRF_FT_STRING                   ThisUpdate;
  TRF_FT_STRING                   NextUpdate;
  TRFID_Items_List               *pExtensions;
  TRFID_Items_List               *pRevokedCertificates;
  TRFID_Items_List               *pNotifications;
  TRF_FT_STRING                   FileName;
  DWORD                           PA_Status;
  TRFID_Certificate_Ex           *pIssuerCertificate;
};
```

Declaration: RFID.h

Fields:

Version – version of TBSCertList ASN.1 structure;

SignatureAlgorithm – CRL digital signature algorithm identifier (OID);

pIssuer – CRL issuer identifier;

ThisUpdate – CRL issue date;

NextUpdate – next CRL release date;

pExtensions – extensions list of CRL. Elements of the list are **TRFID_PKI_Extension ***;

pRevokedCertificates – list of the revoked certificates. Elements of the list are **TRFID_RevocationInfo ***;

pNotifications – list of remarks arisen during the analysis of the data structure of the CRL and performance of its digital signature verification. The elements of the list are DWORD values, corresponding to the constants from **eLDS_ParsingErrorCodes**

FileName – the name of the CRL source file, if there is one (UTF8 string);

PA_Status – result of CRL's digital signature verification (RFID_Error_NotPerformed, RFID_Error_NoError либо `RFID_Error_Session_PA_SignatureCheckFailed`);

`pIssuerCertificate` – reference to the parent certificate.

XML-representation of the structure:

```
<RFID_CRL_Ex>
  <FileName>
  <PA_Status>
  <Version>
  <SignatureAlgorithm>
  <Issuer>
  <ThisUpdate>
  <NextUpdate>
  <Extensions>
     <RFID_PKI_Extension>
     ...
  </Extensions>
  <RevokedCertificates>
     <RFID_RevocationInfo>
     ...
  </RevokedCertificates>
  <Notifications>
     <Item>
     ...
  </Notifications>
</ RFID_CRL_Ex>
```

Values and format of nodes correspond to the fields of **`TRFID_CRL_Ex`** structure. The number of `Extensions` elements corresponds to the number of elements in `pExtensions` array, the number of `RevokedCertificates` elements – to `pRevokedCertificates` array.

`SignatureAlgorithm` element is a string in the format "`S1 (S2)`", where S1 – algorithm name, S2 – identifier (OID string).

`PA_Status` is a string in the format "`S1 [S2]`", where S1 – status code abbreviation, S2 – numeric value in a hexadecimal format.

`Item` elements of `Notifications` list are strings in the format "`S1 [S2]`", where S1 – abbreviation of the corresponding notification code, S2 – the numeric value in a hexadecimal format.

6.3.82. TRFID_AccessKey

`TRFID_AccessKey` structure is used to describe the contents of secure data access key within the context of the communication session with electronic document (see section 5.8.6).

```
struct TRFID_AccessKey
{
```

```
   DWORD          accessType;
   DWORD          keyType;
   TRF_FT_BYTES   AccessKey;
   union
   {
      DWORD  CheckFullKeyMatching;
      DWORD  eSignPIN_Index;
   };
};
```

Declaration: `RFID.h`

Fields:

`accessType`	–	type of secure data access procedure, for which the key is provided (one of **`eRFID_AccessControl_ProcedureType`** values – `acptBAC` or `acptPACE`)
`keyType`	–	key type (one of **`eRFID_Password_Type`** values)
`AccessKey`	–	key contents
`CheckFullKeyMatching`	–	logical sign of the need for a full comparison of `AccessKey` contents with the contents of `DG1` (MRZ) data group
`eSignPIN_Index`	–	used *eSign-PIN* identifier

For the key of `pptPIN_eSign` type the contents of `accessType` are ignored.

XML-representation of the structure:

`<Session_key>`		
`<AccessType>`	–	text abbreviation of `accessType` value
`<KeyType>`	–	text abbreviation of `keyType` value
`<AccessKey>`	–	text `AccessKey` value
`<CheckFullKeyMatching>`	–	boolean `CheckFullKeyMatching` value
`</Session_key>`		

6.3.83. TRFID_Terminal

`TRFID_Terminal` structure is used to describe the terminal type within the context of the communication session with electronic document (see section 5.8.4).

```
struct TRFID_Terminal
{
   DWORD          TermType;
   DWORD          AuthReq;
   DWORD          AuthReq2;
   TRF_FT_BYTES   TermCert_Data;
   char           *TermCert_FileName;
};
```

Declaration: `RFID.h`

Fields:

`TermType`	–	terminal type (one of **`eRFID_TerminalType`** values)
`AuthReq`	–	declared (set) combination of flags of access rights to the functionality of the document (combination of **`eRFID_TerminalAuthorizationRequirement`** values)
`AuthReq2`	–	declared (set) combination of flags of access rights to the functionality of the document (combination of **`eRFID_TerminalAuthorizationRequirement`** values)
`TermCert_Data`	–	terminal certificate binary data
`TermCert_FileName`	–	terminal certificate full file name (in UTF8 format)

The following values from **`eRFID_TerminalAuthorizationRequirement`** are used in the context of `AuthReq2` field:

```
tar_AT_Func_InstallQCert
tar_AT_Func_InstallCert
tar_AT_Func_PINManagement
tar_AT_Func_CAN_Allowed
tar_AT_Func_PrivilegedTerminal
tar_AT_Func_RestrictedIdent
tar_AT_Func_Verify_CommunityID
tar_AT_Func_Verify_Age
tar_AT_Func_Full
```

All the remaining values are used in the context of `AuthReq` field.

XML-representation of the structure:

`<Session_terminal>`

`<TermType>`	–	text abbreviation of `TermType` value
`<AuthReq>`	–	`AuthReq` value in a hexadecimal format (e.g. `"0x00000003"`)
`<AuthReq2>`	–	`AuthReq2` value in a hexadecimal format (e.g. `"0x00000000"`)

`</Session_terminal>`

6.3.84. TRFID_eSignKeyParameters

`TRFID_eSignKeyParameters` structure is used to identify the cryptographic key of *`eSign`* application within the context of the communication session with electronic document (see section 5.8.21).

```
struct TRFID_eSignKeyParameters
{
  BYTE key_Id;
};
```

Declaration: `RFID.h`

Fields:

`key_Id` – identifier of cryptographic key pair of `eSign` application.

6.3.85. TRFID_eSignPINParameters

`TRFID_eSignPINParameters` structure is used to describe *eSign-PIN* parameters within the context of the communication session with electronic document (see section 5.8.20).

```
struct TRFID_eSignPINParameters
{
   BYTE PIN_Id;
   char *PIN_new;
};
```

Declaration: `RFID.h`

Fields:

`PIN_Id` – *eSign-PIN* identifier

`PIN_new` – new value of *eSign-PIN* (for commands of change and creation of the key)

6.3.86. TRFID_ApplicationID

`TRFID_ApplicationID` structure is used to store the application identifier and to use it within the context of the communication session with electronic document (see section 5.8.9).

```
struct TRFID_ApplicationID
{
   DWORD  id_length;
   BYTE   id[28];
};
```

Declaration: `RFID.h`

Field:

`id_length` – `id` identifier length

`id` – application identifier

6.3.87. TRFID_FileID

`TRFID_FileID` structure is used to store the file identifier and to use it within the context of the communication session with electronic document (see sections 5.7.4, 5.8.10).

```
struct TRFID_FileID
{
   BYTE   *pID;
   DWORD  nLength;
```

```
    DWORD   id_type;
    DWORD   SM_protected;
    DWORD   FixedLength;
};
```

Declaration: `RFID.h`

Fields:

`pID` – binary array of the file identifier contents

`nLength` – `pID` array length

`id_type` – file identifier type (one of **`eRFID_FileID_Type`** values)

`SM_protected` – sign that access to the file should be performed through a SM-channel

`FixedLength` – fixed file length if it is known in advance or reading of specific number of bytes is required (`0` – file length is defined automatically by the length of the header tag of its ASN.1 contents)

6.3.88. TRFID_FilesList

`TRFID_FilesList` structure is used to store a list of file identifiers (see section 5.7.4).

```
struct TRFID_FilesList
{
    TRFID_FileID   files[32];
    DWORD          N;
};
```

Declaration: `RFID.h`

Field:

`files` – array of file identifiers

`N` – number of `files` significant elements

6.3.89. TRFID_FileUpdateData

`TRFID_FileUpdateData` structure is used to store the identifier and the new contents of the file for the file updating operation (see section 5.8.19).

```
struct TRFID_FileUpdateData
{
    TRFID_FileID     FileID;
    TCustomRawData   Data;
};
```

Declaration: `RFID.h`

Field:

`FileID` – file identifier

Data – new file contents

6.3.90. TRFID_AccessControl_Params

TRFID_AccessControl_Params structure is used to transfer the parameters of authentication or secure file access procedure within the context of the communication session with electronic document (see section 5.8.7).

```
struct TRFID_AccessControl_Params
{
   DWORD   ac_Type;
   void    *ac_Params;
};
```

Declaration: RFID.h

Fields:

ac_Type – type of procedure (one of **eRFID_AccessControl_ProcedureType** values)

ac_Params – procedure parameters

6.3.91. TTerminalAuthenticationStepData

TTerminalAuthenticationStepData procedure is used to define the parameters of another step of terminal authentication procedure when performing it in step-by-step mode (see section 5.8.15).

```
struct TTerminalAuthenticationStepData
{
   DWORD          step;
   char           CAR[32];
   TRF_FT_BYTES   CVCA_Link_Certificate;
   TRF_FT_BYTES   DV_Certificate
   TRF_FT_BYTES   IS_Certificate;
   TRF_FT_BYTES   IS_PrivateKey;
   TRF_FT_BYTES   Challenge;
   TRF_FT_BYTES   Signature;
};
```

Declaration: RFID.h

Fields:

step – procedure step identifier (1 and 2 values are allowed)

CAR – required CVCA-key identifier (CAR)

CVCA_Link_Certificate – user-defined CVCA-link certificate

DV_Certificate – user-defined DV-certificate

IS_Certificate – user-defined terminal certificate

IS_PrivateKey – private key corresponding to the terminal certificate

`Challenge` – data to be signed or its hash value (data type specified in `Challenge.nType` – one of the value from **`eRFID_TerminalAuthenticationToSignDataType`**);

`Signature` – digital signature for transfer to the RFID-chip

6.3.92. TTerminalVerificationData

`TTerminalVerificationData` structure is used to store information about verified auxiliary data within the context of TA (see section 5.8.18).

```
struct TTerminalVerificationData
{
   TRF_FT_BYTES   DateOfExpiry;
   TRF_FT_BYTES   DateOfBirth;
   TRF_FT_BYTES   CommunityID;
};
```

Declaration: `RFID.h`

Fields:

`DateOfExpiry` – date of document expiry

`DateOfBirth` – date of birth

`CommunityID` – binary array of `Community ID` contents

XML-representation of the structure:

```
<VerifiedData>
  <DateOfExpiry>
     <Value>
     <Status>
  </DateOfExpiry>
  <DateOfBirth>
     <Value>
     <Status>
  </DateOfBirth>
  <CommunityID>
     <Value>
     <Status>
  </CommunityID>
</VerifiedData>
```

The value of `DateOfExpiry` and `DateOfBirth` elements represented as text content of the corresponding structure fields.

The value of `CommunityID` element is a string. Each byte of `CommunityID` represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"A0 00 00 02 47 10 01"`)

`Status` element is a string in the format "`S1 [S2]`", where S1 – status code abbreviation (`nType` value of the corresponding structure element), S2 – numeric value in a hexadecimal format.

6.3.93. TPACE_SetupParams

`TPACE_SetupParams` structure is used to define parameters of PACE procedure within the context of the communication session with electronic document (see section 5.8.8).

```
struct TPACE_SetupParams
{
  DWORD  nOptionIdx;
  BOOL   skipCHAT;
};
```

Declaration: `RFID.h`

Fields:

`nOptionIdx` – index of procedure variant

`skipCHAT` – sign to transmit CHAT when initializing the procedure

6.3.94. TCA_SetupParams

`TCA_SetupParams` structure is used to define parameters of CA procedure within the context of the communication session with electronic document (see section 5.8.14).

```
struct TCA_SetupParams
{
  DWORD  nOptionIdx;
  BOOL   TA_preliminary_step;
};
```

Declaration: `RFID.h`

Fields:

`nOptionIdx` – index of procedure variant

`TA_preliminary_step` – indication of TA preliminary phase performance (`true` or `false`)

6.3.95. TTA_SetupParams

`TTA_SetupParams` structure is used to define parameters of TA procedure within the frames of the current communication session with electronic document (see section 5.8.15).

```
struct TTA_SetupParams
{
```

```
   DWORD   nOptionIdx;
   DWORD   ProcessType;
   TTerminalAuthenticationStepData  TA_StepData;
   TTerminalVerificationData        VerificationData;
};
```

Declaration: `RFID.h`

Fields:

`nOptionIdx`	–	index of procedure variant
`ProcessType`	–	order of procedure performance (one of **`eRFID_TerminalAuthenticationType`** values)
`TA_StepData`	–	configuration of the next TA step when working in step-by-step mode without using the callback-function
`VerificationData`	–	contents of the auxiliary data for the following verification (see section 4.10.2)

6.3.96. TPA_Params

`TPA_Params` structure is used to define parameters of passive authentication (SO verification) within the frames of the current communication session with electronic document (see section 5.8.12).

```
struct TPA_Params
{
   DWORD   SO_Index;
   DWORD   SI_Index;
   TRF_FT_BYTES   CSCA_Certificate;
   TRF_FT_BYTES   DS_Certificate;
};
```

Declaration: `RFID.h`

Fields:

`SO_Index`	–	index of the verified SO
`SO_Index`	–	index of the verified digital signature
`CSCA_Certificate`	–	user-defined CSCA-certificate
`DS_Certificate`	–	user-defined DS-certificate

6.3.97. TRI_SetupParams

`TRI_SetupParams` structure is used to define parameters of RI procedure within the frames of the current communication session with electronic document (see section 5.8.17).

```
struct TRI_SetupParams
{
   DWORD          nOptionIdx;
   TRF_FT_BYTES   SectorKey1;
```

```
   char              *SectorKey1_FileName;
   TRF_FT_BYTES      SectorKey2;
   char              *SectorKey2_FileName;
};
```

Declaration: `RFID.h`

Field:

`nOptionIdx` – index of procedure variant

`SectorKey1` – data of the public key `1`

`SectorKey1_FileName` – full file name of the public key `1` (in UTF8 format)

`SectorKey2` – data of the public key `2`

`SectorKey2_FileName` – full file name of the public key `2` (in UTF8 format)

6.3.98. TRF_EDL_DG1

`TRF_EDL_DG1` structure is used to store the contents of informational `EF.DG1` data group of *`eDL`* application – mandatory demographic data and vehicle categories/restrictions/conditions [39].

```
struct TRF_EDL_DG1
{
  BYTE              nType;
  TRF_FT_STRING     Surname;
  TRF_FT_STRING     GivenNames;
  TRF_FT_STRING     DateOfBirth;
  TRF_FT_STRING     PlaceOfBirth;
  TRF_FT_STRING     Nationality;
  TRF_FT_STRING     Gender;
  TRF_FT_STRING     IssuingCountry;
  TRF_FT_STRING     DateOfIssue;
  TRF_FT_STRING     DateOfExpiry;
  TRF_FT_STRING     IssuingAuthority;
  TRF_FT_STRING     AdminNumber;
  TRF_FT_STRING     DocumentNumber;
  TRF_FT_STRING     Address;
  BYTE              nVRCRecords;
  TRF_FT_BYTES    **pVRCContents;
};
```

Declaration: `RFID.h`

Fields:

`nType` – type of informational data group; always contains `RFDGT_EDL_DG1` value from **`eRFID_DataGroupTypeTag`** enumeration;

`Surname` – surname(s) of the holder;

`GivenNames` – other name(s) of the holder;

`DateOfBirth` – date of birth;

`PlaceOfBirth` – place of birth;

`Nationality` – nationality;

`Gender`	– gender;
`IssuingCountry`	– issuing state;
`DateOfIssue`	– date of issue of the licence;
`DateOfExpiry`	– date of expiry of the licence;
`IssuingAuthority`	– issuing authority;
`AdminNumber`	– administrative number (other than document number);
`DocumentNumber`	– document number;
`Address`	– permanent place of residence or postal address;
`nVRCRecords`	– number of `pVRCContents` elements;
`pVRCContents`	– array of textual vehicle categories/restrictions/conditions elements.

XML-representation of the structure:

```
<eDL_DG1>
  <Type>
  <Surname>
  <GivenNames>
  <DateOfBirth>
  <PlaceOfBirth>
  <Nationality>
  <Gender>
  <IssuingCountry>
  <DateOfIssue>
  <DateOfExpiry>
  <IssuingAuthority>
  <AdminNumber>
  <DocumentNumber>
  <Address>
  <VRCContents>
     <VRCRecord>
     ...
  </VRCContents>
</eDL_DG1>
```

Values and format of nodes correspond to the fields of **`TRF_EDL_DG1`** structure.

6.3.99. TRFChipProperties

`TRFChipProperties` structure is used to store information about the characteristics of the RFID-chip located in the scope of the reader (see sections 5.7.1, 5.8.3) [18, 19]. Available when working with readers with firmware version 21.00 and higher.

```
struct TRFChipProperties
{
  BYTE   Type;
  BYTE   Support_4;
  BYTE   Support_DS;
  BYTE   Support_DR;
  BYTE   Actual_DS;
```

```
  BYTE   Actual_DR;
  WORD   FSC;
  BYTE   SFGI;
  BYTE   NAD;
  BYTE   CID;
  BYTE   FWI;
  DWORD  MBL;
  BYTE   SizeUID;
  BYTE   UID[10];
  BYTE   SizeHBytes;
  BYTE   HBytes[16];
  BYTE   SizeATR;
  BYTE   ATR[36];
  BYTE   Support_Mifare;
  BYTE   SAK;
};
```

Declaration: `RFID.h`

Fields:

`Type` – type of the RFID-chip by the connection physical parameters (one of **`eRFID_Type`** constants);

`Support_4` – sign of support for ISO/IEC 14443-4 data exchange protocol – `true` or `false`;

`Support_DS` – combination of **`eRFID_BaudRate`** flags, defining the data transmitting rates supported by the RFID-chip;

`Support_DR` – combination of **`eRFID_BaudRate`** flags, defining the data receiving rates supported by the RFID-chip;

`Actual_DS` – **`eRFID_BaudRate`** value, indicating the established rate for data transmitting to the RFID-chip;

`Actual_DR` – **`eRFID_BaudRate`** value, indicating the established rate for data receiving from the RFID-chip;

`FSC` – size of RFID-chip's receiving buffer for one operation of data transfer (in bytes) (*Frame Size Card*);

`SFGI` – indicator of the minimum time of RFID-chip readiness to receive data from the reader after the end of its own data transmission (*Start-up Frame Guard Time indicator*);

`NAD` – sign of NAD support– `true` или `false`;

`CID` – sign of CID support – `true` или `false`;

`FWI` – indicator of the maximum waiting time for the arrival of data from RFID chip in response to the command sent (*Frame Waiting Time indicator*);

`MBL` – maximum size of type-B RFID-chip data receiving buffer (in bytes) (*Maximum Buffer Length*);

`SizeUID` – length of `UID` field;

`UID` – unique chip identifier;

`SizeHBytes` – length of `HBytes` field;

`HBytes` – historical bytes from the response of type-A RFID-chip to RATS command of ISO/IEC 14443-3 protocol;

`SizeATR` – length of `ATR` field;

`ATR` – ATR string of the chip;

`Support_Mifare` – sign of support for ISO/IEC 14443-3 data exchange protocol (MIFARE® Classic Protocol) – `true` or `false`;

`SAK` – response of type-A RFID-chip to SELECT command of ISO/IEC 14443-3 protocol (*Select Acknowledge*, *SAK*).

XML-representation of the structure:

`<CardProperties2>`

`<Type>`	– text abbreviation of `Type` value
`<Support_4>`	– logical `Support_4` value
`<Support_DS>`	– numeric `Support_DS` value in hexadecimal format (e.g. `"0x0F"`)
`<Support_DR>`	– numeric `Support_DR` value in hexadecimal format (e.g. `"0x0F"`)
`<Actual_DS>`	– numeric `Actual_DS` value in hexadecimal format (e.g. `"0x0F"`)
`<Actual_DR>`	– numeric `Actual_DR` value in hexadecimal format (e.g. `"0x0F"`)
`<FSC>`	– numeric `FSC` value
`<SFGI>`	– numeric `SFGI` value in hexadecimal format (e.g. `"0x01"`)
`<NAD>`	– logical `NAD` value
`<CID>`	– logical `CID` value
`<FWI>`	– numeric `FWI` value in hexadecimal format (e.g. `"0x08"`)
`<MBL>`	– numeric `MBL` value
`<UID>`	– `UID` contents in text format. Each byte is represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"F9 4F 41 60"`)
`<HBytes>`	– `HBytes` contents in text format. Each byte is represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"80 91 E1 31 D8 65 B2 8C 01 01 0E 73 C4 41 E0"`)
`<ATR>`	– `ATR` contents in text format. Each byte is represented by its hexadecimal value. The individual bytes are separated by spaces (e.g. `"3B 8F 80 01 80 91 E1 31 D8 65 B2 8C 01 01 0E 73 C4 41 E0 54"`)
`<Support_Mifare>`	– logical `Support_Mifare` value
`<SAK>`	– numeric `SAK` value in hexadecimal format (e.g. `"0x00"`)

`</CardProperties2>`

6.4. ENUMERATIONS

6.4.1. eRFID_ResultType

eRFID_ResultType enumeration contains a set of constants specifying the type of data stored in **TResultContainer** container structure (see section 6.3.2).

```
enum eRFID_ResultType
{
   RFID_ResultType_Empty                  = 0,
   RFID_ResultType_RFID_RawData           = 101,
   RFID_ResultType_RFID_TextData          = 102,
   RFID_ResultType_RFID_ImageData         = 103,
   RFID_ResultType_RFID_BinaryData        = 104,
   RFID_ResultType_RFID_OriginalGraphics= 105,
};
```

6.4.2. eRFID_DataGroups

eRFID_DataGroups enumeration contains a set of constants specifying the informational data groups, the contents of which needs to be obtained when executing the reading command when working in batch mode (see section 5.7.4).

```
enum eRFID_DataGroups
{
   RFDG_DG1    = 0x00000001,
   RFDG_DG2    = 0x00000002,
   RFDG_DG3    = 0x00000004,
   RFDG_DG4    = 0x00000008,
   RFDG_DG5    = 0x00000010,
   RFDG_DG6    = 0x00000020,
   RFDG_DG7    = 0x00000040,
   RFDG_DG8    = 0x00000080,
   RFDG_DG9    = 0x00000100,
   RFDG_DG10   = 0x00000200,
   RFDG_DG11   = 0x00000400,
   RFDG_DG12   = 0x00000800,
   RFDG_DG13   = 0x00001000,
   RFDG_DG14   = 0x00002000,
   RFDG_DG15   = 0x00004000,
   RFDG_DG16   = 0x00008000,
   RFDG_SOD    = 0x00010000,
   RFDG_USER   = 0x00100000,
   RFDG_DG_All = 0xffffffff,
};
```

Constants correspond to the informational data groups of *ePassport* application, as well as:

RFDG_USER	–	user-defined files from the list assigned by **RFID_Command_SetUserDefinedFilesToRead** command;
RFDG_DG_ALL	–	combination of all available data groups.

6.4.3. eRFID_DataGroupTypeTag

eRFID_DataGroupTypeTag enumeration contains a set of constants specifying the identifiers (ASN.1-tags) of the informational data groups.

```
enum eRFID_DataGroupTypeTag
{
   RFDGT_COM        = 0x60,
   RFDGT_DG1        = 0x61,
   RFDGT_DG2        = 0x75,
   RFDGT_DG3        = 0x63,
   RFDGT_DG4        = 0x76,
   RFDGT_DG5        = 0x65,
   RFDGT_DG6        = 0x66,
   RFDGT_DG7        = 0x67,
   RFDGT_DG8        = 0x68,
   RFDGT_DG9        = 0x69,
   RFDGT_DG10       = 0x6A,
   RFDGT_DG11       = 0x6B,
   RFDGT_DG12       = 0x6C,
   RFDGT_DG13       = 0x6D,
   RFDGT_DG14       = 0x6E,
   RFDGT_DG15       = 0x6F,
   RFDGT_DG16       = 0x70,
   RFDGT_SOD        = 0x77,
   RFDGT_EID_DG1    = 0x61,
   RFDGT_EID_DG2    = 0x62,
   RFDGT_EID_DG3    = 0x63,
   RFDGT_EID_DG4    = 0x64,
   RFDGT_EID_DG5    = 0x65,
   RFDGT_EID_DG6    = 0x66,
   RFDGT_EID_DG7    = 0x67,
   RFDGT_EID_DG8    = 0x68,
   RFDGT_EID_DG9    = 0x69,
   RFDGT_EID_DG10   = 0x6A,
   RFDGT_EID_DG11   = 0x6B,
   RFDGT_EID_DG12   = 0x6C,
   RFDGT_EID_DG13   = 0x6D,
   RFDGT_EID_DG14   = 0x6E,
   RFDGT_EID_DG15   = 0x6F,
   RFDGT_EID_DG16   = 0x70,
   RFDGT_EID_DG17   = 0x71,
   RFDGT_EID_DG18   = 0x72,
   RFDGT_EID_DG19   = 0x73,
   RFDGT_EID_DG20   = 0x74,
   RFDGT_EID_DG21   = 0x75,
   RFDGT_EDL_COM    = 0x60,
   RFDGT_EDL_SOD    = 0x77,
   RFDGT_EDL_CE     = 0x53,
   RFDGT_EDL_DG1    = 0x61,
   RFDGT_EDL_DG2    = 0x6B,
   RFDGT_EDL_DG3    = 0x6C,
   RFDGT_EDL_DG4    = 0x65,
   RFDGT_EDL_DG5    = 0x67,
   RFDGT_EDL_DG6    = 0x75,
   RFDGT_EDL_DG7    = 0x63,
```

```
   RFDGT_EDL_DG8   = 0x76,
   RFDGT_EDL_DG9   = 0x70,
   RFDGT_EDL_DG11  = 0x6D,
   RFDGT_EDL_DG12  = 0x71,
   RFDGT_EDL_DG13  = 0x6F,
   RFDGT_EDL_DG14  = 0x6E,
};
```

Constants with prefix `RFDGT_` correspond to the informational data groups of *`ePassport`* application, with prefix `RFDGT_EID_` – those of *`eID`* application,, with prefix `RFDGT_EDL_` – *`eDL`* application.

6.4.4. eRFID_Type

`eRFID_Type` enumeration contains a set of constants specifying the type of the RFID-chip by the physical parameters of connection between antennas of the chip and the reader (see section 6.3.13).

```
enum eRFID_Type
{
   rftTypeUnknown   = 0,
   rftTypeA         = 1,
   rftTypeB         = 2,
};
```

Value of constants of RFID-chip type:

`rftTypeUnknown` – unknown;
`rftTypeA` – type «A»;
`rftTypeB` – type «B».

6.4.5. eRFID_A_Chip

`eRFID_A_Chip` enumeration contains a set of constants specifying the type of the RFID-chip from MIFARE® family (for chips of type «A») (see section 6.3.13).

```
enum eRFID_A_Chip
{
   rfacUnknown            = 0,
   rfacMifare1K           = 1,
   rfacMifare4K           = 2,
   rfacMifareUltralight   = 3,
   rfacMifareDESFire      = 4,
   rfacMifareProX         = 5,
};
```

Value of chip type constants:

`rfacUnknown` – unknown;
`rfacMifare1K` – MIFARE® 1K;
`rfacMifare4K` – MIFARE® 4K;
`rfacMifareUltralight` – MIFARE® Ultralight;

`rfacMifareDESFire` – MIFARE® DESFire;

`rfacMifareProX` – MIFARE® ProX or SmartMX xD(T).

6.4.6. eRFID_BaudRate

eRFID_BaudRate enumeration contains a set of constants specifying the rate of data exchange between the reader and the RFID-chip (see section 6.3.13).

```
enum eRFID_BaudRate
{
   rfbr_106 = 0x01,
   rfbr_212 = 0x02,
   rfbr_424 = 0x04,
   rfbr_848 = 0x08,
};
```

Value of constants of data exchange rate:

`rfbr_106` – 106 bits/s;

`rfbr_212` – 212 bits/s;

`rfbr_424` – 424 bits/s;

`rfbr_848` – 848 bits/s.

6.4.7. eCBEFF_Gender

eCBEFF_Gender enumeration contains a set of constants specifying the sex from the record of biometric graphic data of the document owner (see section 6.3.25).

```
enum eCBEFF_Gender
{
   gndrUnspecified  = 0,
   gndrMale         = 1,
   gndrFemale       = 2,
   gndrUnknown      = 0xff,
};
```

Value of constants of sex:

`gndrUnspecified` – unspecified;

`gndrMale` – male;

`gndrFemale` – female;

`gndrUnknown` – unknown.

6.4.8. eCBEFF_EyeColor

eCBEFF_EyeColor enumeration contains a set of constants specifying the eye color from the record of biometric graphic data of the document owner (see section 6.3.25).

```
enum eCBEFF_EyeColor
{
   eyeUnspecified   = 0,
   eyeBlack         = 1,
   eyeBlue          = 2,
   eyeBrown         = 3,
   eyeGray          = 4,
   eyeGreen         = 5,
   eyeMultiColored  = 6,
   eyePink          = 7,
   eyeOther         = 0xff,
};
```

Value of constants of eye color:

`eyeUnspecified`	–	unspecified;
`eyeBlack`	–	black;
`eyeBlue`	–	blue;
`eyeBrown`	–	brown;
`eyeGray`	–	gray;
`eyeGreen`	–	green;
`eyeMultiColored`	–	multi colored;
`eyePink`	–	pink;
`eyeOther`	–	other.

6.4.9. eCBEFF_HairColor

eCBEFF_HairColor enumeration contains a set of constants specifying the hair color from the record of biometric graphic data of the document owner (see section 6.3.25).

```
enum eCBEFF_HairColor
{
   hairUnspecified  = 0,
   hairBald         = 1,
   hairBlack        = 2,
   hairBlonde       = 3,
   hairBrown        = 4,
   hairGray         = 5,
   hairWhite        = 6,
   hairRed          = 7,
   hairOther        = 0xff,
};
```

Value of constants of hair color:

`hairUnspecified`	–	unspecified;
`hairBald`	–	bald;
`hairBlack`	–	black;
`hairBlonde`	–	blonde;
`hairBrown`	–	brown;

`hairGray` – gray;
`hairWhite` – white;
`hairRed` – red;
`hairOther` – other.

6.4.10. eCBEFF_FaceFeatureMask

`eCBEFF_FaceFeatureMask` enumeration contains a set of masks for determination of the presence of additional features in the record of biometric facial graphic data of the document owner (see section 6.3.25).

```
enum eCBEFF_FaceFeatureMask
{
   ffmGlasses             = 0x000003,
   ffmMoustache           = 0x000005,
   ffmBeard               = 0x000009,
   ffmTeethVisible        = 0x000011,
   ffmBlink               = 0x000021,
   ffmMouthOpen           = 0x000041,
   ffmLeftEyePatch        = 0x000081,
   ffmRightEyePatch       = 0x000101,
   ffmDarkGlasses         = 0x000201,
   ffmDistortionMedical   = 0x000401,
};
```

Value of constants of additional features:

`ffmGlasses` – glasses;
`ffmMoustache` – moustache;
`ffmBeard` – beard;
`ffmTeethVisible` – teeth visible;
`ffmBlink` – blinking;
`ffmMouthOpen` – mouth open;
`ffmLeftEyePatch` – left eye patch;
`ffmRightEyePatch` – right eye patch;
`ffmDarkGlasses` – dark glasses;
`ffmDistortionMedical` – face is distorted due to medical reasons.

6.4.11. eCBEFF_FaceExpression

`eCBEFF_FaceExpression` enumeration contains a set of constants specifying the facial expression in the record of biometric facial graphic data of the document owner (see section 6.3.25).

```
enum eCBEFF_FaceExpression
{
   feUnspecified       = 0x0000,
   feNeutral           = 0x0001,
```

```
   feSmile1          = 0x0002,
   feSmile2          = 0x0003,
   feRaisedEyebrows  = 0x0004,
   feEyesLookingAway = 0x0005,
   feSquinting       = 0x0006,
   feFrowning        = 0x0007,
};
```

Constants correspond to various facial expressions:

`feUnspecified` – unspecified;
`feNeutral` – neutral;
`feSmile1` – smile 1;
`feSmile2` – smile 2;
`feRaisedEyebrows` – raised eyebrows;
`feEyesLookingAway` – eyes looking away;
`feSquinting` – squinting;
`feFrowning` – frowning.

6.4.12. eCBEFF_FaceImageType

eCBEFF_FaceImageType enumeration contains a set of constants specifying the type of image in the record containing biometric facial graphic data of the document owner (see section 6.3.28) (in compliance with the ISO/IEC FCD 19794-5:2003).

```
enum eCBEFF_FaceImageType
{
   fitUnspecified   = 0,
   fitBasic         = 1,
   fitFullFrontal   = 2,
   fitTokenFrontal  = 3,
   fitOther         = 4,
};
```

Constants correspond to various image types:

`fitUnspecified` – unspecified;
`fitBasic` – basic;
`fitFullFrontal` – full frontal;
`fitTokenFrontal` – partially frontal;
`fitOther` – other.

6.4.13. eCBEFF_FaceImageTypeFDIS

eCBEFF_FaceImageTypeFDIS enumeration contains a set of constants specifying the type of image in the record of biometric facial graphic data of the document owner (see section 6.3.28) (in compliance with the ISO/IEC 19794-5:2005).

```
enum eCBEFF_FaceImageTypeFDIS
{
   fitFDISBasic          = 0,
   fitFDISFullFrontal    = 1,
   fitFDISTokenFrontal   = 2,
};
```

Constants correspond to various image types:

`fitFDISBasic`	–	basic;
`fitFDISFullFrontal`	–	full frontal;
`fitFDISTokenFrontal`	–	partially frontal.

6.4.14. eCBEFF_ImageDataType

`eCBEFF_ImageDataType` enumeration contains a set of constants specifying the format of image data in the record of biometric facial graphic data of the document owner (see section 6.3.28).

```
enum eCBEFF_ImageDataType
{
   idtJPEG                = 0,
   idtJPEG2000            = 1,
   idtJPEG2000Lossless    = 2,
   idtPNG                 = 3,
};
```

Value of constants of image data format:

`idtJPEG`	–	JPEG;
`idtJPEG2000`	–	JPEG-2000;
`idtJPEG2000Lossless`	–	lossless JPEG;
`idtPNG`	–	PNG.

6.4.15. eCBEFF_ImageColorSpace

`eCBEFF_ImageColorSpace` enumeration contains a set of constants specifying the image color space in the record of biometric facial graphic data of the document owner (see section 6.3.28).

```
enum eCBEFF_ImageColorSpace
{
   icsUnspecified   = 0,
   ics24BitRGB      = 1,
   icsYUV422        = 2,
   ics8BitGrayscale = 3,
   icsOther         = 4,
};
```

Value of constants of image color space:

`icsUnspecified`	–	unspecified;

`ics24BitRGB` – 24 bits per color;
`icsYUV422` – YUV 4:2:2;
`ics8BitGrayscale` – grayscale;
`icsOther` – other.

6.4.16. eCBEFF_ImageSourceType

eCBEFF_ImageSourceType enumeration contains a set of constants specifying the source of the captured image in the record of biometric facial graphic data of the document owner (see section 6.3.28).

```
enum eCBEFF_ImageSourceType
{
   istUnspecified                = 0,
   istPhotoUnknown               = 1,
   istPhotoDigitalCamera         = 2,
   istPhotoScanner               = 3,
   istVideoFrameUnknown          = 4,
   istVideoFrameAnalogueCamera   = 5,
   istVideoFrameDigitalCamera    = 6,
   istUnknown                    = 7,
};
```

Constants correspond to various sources of captured image:

`istUnspecified` – unspecified;
`istPhotoUnknown` – photo;
`istPhotoDigitalCamera` – photo (digital camera);
`istPhotoScanner` – photo (scanner);
`istVideoFrameUnknown` – video frame;
`istVideoFrameAnalogueCamera` – video frame (analogue camera);
`istVideoFrameDigitalCamera` – video frame (digital camera);
`istUnknown` – unknown.

6.4.17. eCBEFF_BiometricType

eCBEFF_BiometricType enumeration contains a set of constants specifying the type of biometric data stored in the record of informational data group of the document (see section 6.3.22).

```
enum eCBEFF_BiometricType
{
   btUnknown        = 0x000000,
   btMultiple       = 0x000001,
   btFacial         = 0x000002,
   btVoice          = 0x000004,
   btFingerPrint    = 0x000008,
   btIris           = 0x000010,
   btRetina         = 0x000020,
   btHandGeometry   = 0x000040,
```

```
  btSignature      = 0x000080,
  btKeystroke      = 0x000100,
  btLipMovement    = 0x000200,
  btThermalFace    = 0x000400,
  btThermalHand    = 0x000800,
  btGait           = 0x001000,
  btBodyOdor       = 0x002000,
  btDNA            = 0x004000,
  btEarShape       = 0x008000,
  btFingerGeometry = 0x010000,
  btPalmPrint      = 0x020000,
  btVeinPattern    = 0x040000,
  btFootPrint      = 0x080000,
};
```

Value of constants of biometric data types:

`btUnknown` – unknown biometric data;
`btMultiple` – combined biometric data;
`btFacial` – face;
`btVoice` – voice;
`btFingerPrint` – fingerprint;
`btIris` – iris;
`btRetina` – retina;
`btHandGeometry` – hand geometry data;
`btSignature` – signature;
`btKeystroke` – handwriting;
`btLipMovement` – data on lip movement;
`btThermalFace` – thermal face map;
`btThermalHand` – thermal hand map;
`btGait` – data on gait;
`btBodyOdor` – body odor;
`btDNA` – DNA;
`btEarShape` – ear shape;
`btFingerGeometry` – finger geometry;
`btPalmPrint` – palm print;
`btVeinPattern` – vein pattern;
`btFootPrint` – foot print.

6.4.18. eCBEFF_BiometricSubTypeMask

eCBEFF_BiometricSubTypeMask enumeration contains a set of constants specifying the subtype of biometric data stored in the record of informational data group of the document (see section 6.3.22).

```
enum eCBEFF_BiometricSubTypeMask
{
  bstMaskRight           = 0x01,
```

```
   bstMaskLeft            = 0x02,
   bstMaskThumb           = 0x04,
   bstMaskPointerFinger   = 0x08,
   bstMaskMiddleFinger    = 0x0C,
   bstMaskRingFinger      = 0x10,
   bstMaskLittleFinger    = 0x14,
};
```

Value of the constants:

`bstMaskRight`	–	right (general feature);
`bstMaskLeft`	–	left (general feature);
`bstMaskThumb`	–	thumb;
`bstMaskPointerFinger`	–	index finger;
`bstMaskMiddleFinger`	–	middle finger;
`bstMaskRingFinger`	–	ring finger;
`bstMaskLittleFinger`	–	little finger.

6.4.19. eCBEFF_FormatOwners

eCBEFF_FormatOwners enumeration contains a set of constants specifying the identifier of the format owner of biometric data representation (see section 6.3.22).

```
enum eCBEFF_FormatOwners
{
   fownUndefined            = 0,
   fownISO_IEC_JTC_1_SC_37 = 0x0101,
};
```

6.4.20. eBIT_SecurityOptions

eBIT_SecurityOptions enumeration contains a set of constants specifying the parameters of biometric data record protection (see section 6.3.22).

```
enum eBIT_SecurityOptions
{
   scoNotDefined          = -1,
   scoNone                =  0,
   scoPrivacy             =  1,
   scoIntegrity           =  2,
   scoProvacuIntegrity    =  3,
};
```

6.4.21. eBIT_IntegrityOptions

eBIT_IntegrityOptions enumeration contains a set of constants specifying the parameters of biometric data record integrity (see section 6.3.22).

```
enum eBIT_IntegrityOptions
```

```
{
   itoNotDefined = -1,
   itoNone       =  0,
   itoMAC        =  1,
   itoSigned     =  2,
};
```

6.4.22. eCBEFF_FormatTypes

eCBEFF_FormatTypes enumeration contains a set of constants specifying the identifier of the biometric data record format (see section 6.3.22).

```
enum eCBEFF_FormatTypes
{
   ftypFinger_Minutiae                        = 0x0201,
   ftypFinger_MinutiaeExtended                = 0x0202,
   ftypFinger_Pattern                         = 0x0301,
   ftypFinger_PatternExtended                 = 0x0302,
   ftypFinger_Image                           = 0x0401,
   ftypFace_Image                             = 0x0501,
   ftypIris_Image                             = 0x0601,
   ftypIris_ImageExtended                     = 0x0602,
   ftypSignatureRaw                           = 0x0701,
   ftypSignatureRawExtended                   = 0x0702,
   ftypSignatureCommonFeature                 = 0x0703,
   ftypSignatureCommonFeatureExtended         = 0x0704,
   ftypSignatureBoth                          = 0x0705,
   ftypSignatureBothExtended                  = 0x0706,
   ftypHandGeometry                           = 0x0801,
   ftypHandGeometryExtended                   = 0x0802,
   ftypFinger_Minutiae_FDIS                   = 0x0001,
   ftypFinger_MinutiaeExtended_FDIS           = 0x0002,
   ftypFinger_Image_FDIS                      = 0x0007,
   ftypFace_Image_FDIS                        = 0x0008,
   ftypIris_Image_FDIS                        = 0x0009,
   ftypIris_ImageExtended_FDIS                = 0x000B
};
```

Value of constants:

`ftypFinger_Minutiae` – minutiae data;
`ftypFinger_MinutiaeExtended` – extended minutiae data;
`ftypFinger_Pattern` – fingerprint template;
`ftypFinger_PatternExtended` – extended template of fingerprints;
`ftypFinger_Image` – fingerprints image;
`ftypFace_Image` – face image;
`ftypIris_Image` – iris image;
`ftypIris_ImageExtended` – extended iris image;
`ftypSignatureRaw` – signature image;

`ftypSignatureRawExtended` – extended signature image;
`ftypSignatureCommonFeature` – common features of signature;
`ftypSignatureCommonFeatureExtended` – extended common features of signature;
`ftypSignatureBoth` – common features of signature;
`ftypSignatureBothExtended` – extended common features of signature;
`ftypHandGeometry` – hand geometry;
`ftypHandGeometryExtended` – extended hand geometry;
`ftypFinger_Minutiae_FDIS` – minutiae data (in compliance with the ISO/IEC 19794-5:2005);
`ftypFinger_MinutiaeExtended_FDIS` – extended minutiae data (in compliance with the ISO/IEC 19794-5:2005);
`ftypFinger_Image_FDIS` – fingerprint image (in compliance with the ISO/IEC 19794-5:2005);
`ftypFace_Image_FDIS` – face image (in compliance with the ISO/IEC 19794-5:2005);
`ftypIris_Image_FDIS` – iris image (in compliance with the ISO/IEC 19794-5:2005);
`ftypIris_ImageExtended_FDIS` – extended iris image (in compliance with the ISO/IEC 19794-5:2005).

6.4.23. eCBEFF_ImageCompressionAlgorithm

eCBEFF_ImageCompressionAlgorithm enumeration contains a set of constants specifying the format of image data in the record of biometric graphic data of fingerprints (palms) of the document owner (see section 6.3.29).

```
enum eCBEFF_ImageCompressionAlgorithm
{
   icaUncompressedNoBitPacking  = 0,
   icaUncompressedBitPacked     = 1,
   icaCompressedWSQ             = 2,
   icaCompressedJPEG            = 3,
   icaCompressedJPEG2000        = 4,
   icaCompressedPNG             = 5,
};
```

Value of constants of the format:

`icaUncompressedNoBitPacking` – without compression;
`icaUncompressedBitPacked` – bit-packed;
`icaCompressedWSQ` – WSQ;
`icaCompressedJPEG` – JPEG;
`icaCompressedJPEG2000` – JPEG-2000;
`icaCompressedPNG` – PNG.

6.4.24. eCBEFF_FingerPalmPosition

eCBEFF_FingerPalmPosition enumeration contains a set of constants specifying the finger (palm) position for the current template from the biometric data record (see section 6.3.30).

```
enum eCBEFF_FingerPalmPosition
{
   fppUnknown             = 0,
   fppRightThumb          = 1,
   fppRightIndexFinger    = 2,
   fppRightMiddleFinger   = 3,
   fppRightRingFinger     = 4,
   fppRightLittleFinger   = 5,
   fppLeftThumb           = 6,
   fppLeftIndexFinger     = 7,
   fppLeftMiddleFinger    = 8,
   fppLeftRingFinger      = 9,
   fppLeftLittleFinger    = 10,
   fppPlainRight4Fingers= 13,
   fppPlainLeft4Fingers   = 14,
   fppPlainThumbs2        = 15,
   fppUnknownPalm         = 20,
   fppRightFullPalm       = 21,
   fppRightWritersPalm    = 22,
   fppLeftFullPalm        = 23,
   fppLeftWritersPalm     = 24,
   fppRightLowerPalm      = 25,
   fppRightUpperPalm      = 26,
   fppLeftLowerPalm       = 27,
   fppLeftUpperPalm       = 28,
   fppRightOther          = 29,
   fppLeftOther           = 30,
   fppRightInterdigital   = 31,
   fppRightThenar         = 32,
   fppRightHypothenar     = 33,
   fppLeftInterdigital    = 34,
   fppLeftThenar          = 35,
   fppLeftHypothenar      = 36,
};
```

Value of constants:

fppUnknown	–	position unknown;
fppRightThumb	–	right thumb;
fppRightIndexFinger	–	right index finger;
fppRightMiddleFinger	–	right middle finger;
fppRightRingFinger	–	right ring finger;
fppRightLittleFinger	–	right little finger;
fppLeftThumb	–	left thumb;
fppLeftIndexFinger	–	left index finger;
fppLeftMiddleFinger	–	left middle finger;

`fppLeftRingFinger` – left ring finger;
`fppLeftLittleFinger` – left little finger;
`fppPlainRight4Fingers` – control 4-fingerprint of the right hand;
`fppPlainLeft4Fingers` – control 4-fingerprint of the left hand;
`fppPlainThumbs2` – control print of thumbs;
`fppUnknownPalm` – unknown palm;
`fppRightFullPalm` – right palm;
`fppRightWritersPalm` – right writer's palm;
`fppLeftFullPalm` – left palm;
`fppLeftWritersPalm` – left writer's palm;
`fppRightLowerPalm` – lower part of the right palm;
`fppRightUpperPalm` – upper part of the right palm;
`fppLeftLowerPalm` – lower part of the left palm;
`fppLeftUpperPalm` – upper part of the left palm;
`fppRightOther` – other right hand print;
`fppLeftOther` – other left hand print;
`fppRightInterdigital` – interdigital of the right hand;
`fppRightThenar` – thenar of the right hand;
`fppRightHypothenar` – hypothenar of the right hand;
`fppLeftInterdigital` – interdigital of the left hand;
`fppLeftThenar` – thenar of the left hand;
`fppLeftHypothenar` – hypo thenar of the left hand.

6.4.25. eCBEFF_FingerPalmImpression

`eCBEFF_FingerPalmImpression` enumeration contains a set of constants specifying the method of acquiring fingerprints for the current template from the record of biometric graphic data of the fingerprints (palm prints) of the document owner (see section 6.3.30).

```
enum eCBEFF_FingerPalmImpression
{
  fpiLiveScanPlain        = 0,
  fpiLiveScanRolled       = 1,
  fpiNonLiveScanPlain     = 2,
  fpiNonLiveScanRolled    = 3,
  fpiLatent               = 4,
  fpiSwipe                = 8,
  fpiLiveScanContactless  = 9,
};
```

Value of constants:

`fpiLiveScanPlain` – "live" scanning;
`fpiLiveScanRolled` – "live" rolling;
`fpiNonLiveScanPlain` – scanning;
`fpiNonLiveScanRolled` – rolling;

`fpiLatent`	–	latent method;
`fpiSwipe`	–	swipe;
`fpiLiveScanContactless`	–	contactless scanning.

6.4.26. eCBEFF_ScaleUnits

`eCBEFF_ScaleUnits` enumeration contains a set of constants specifying the units of resolution for images in the record of biometric graphic data of fingerprints (palms) of the document owner (see section 6.3.29).

```
enum eCBEFF_ScaleUnits
{
   suUnspecified         = 0,
   suPixelsPerInch       = 1,
   suPixelsPerCentimeter= 2,
};
```

Constants correspond to various measurement units:

`suUnspecified`	–	unspecified;
`suPixelsPerInch`	–	pixels per inch (ppi);
`suPixelsPerCentimeter`	–	pixels per centimeter;

6.4.27. eIrisImageProperties

`eIrisImageProperties` enumeration contains a set of constants and the bit masks defining the parameters of images stored in a single record of iris graphic data of the document owner (see section 6.3.41).

```
enum eIrisImageProperties
{
   iipmHorzOrientation_Undefined     = 0x0000,
   iipmHorzOrientation_Base          = 0x0001,
   iipmHorzOrientation_Flipped       = 0x0002,
   iipmHorzOrientation_Mask          = 0x0003,

   iipmVertOrientation_Undefined     = 0x0000,
   iipmVertOrientation_Base          = 0x0004,
   iipmVertOrientation_Flipped       = 0x0008,
   iipmVertOrientation_Mask          = 0x000d,

   //rectilinear only
   iipmScanType_Undefined            = 0x0000,
   iipmScanType_Progressive          = 0x0010,
   iipmScanType_InterlaceFrame       = 0x0020,
   iipmScanType_InterlaceField       = 0x0030,
   iipmScanType_Mask                 = 0x0030,

   //all values below - polar only
```

```
   iipmOcclusions_Undefined          = 0x0000,
   iipmOcclusions_Processed          = 0x0040,
   iipmOcclusions_Mask               = 0x0040,

   iipmOcclusionFilling_ZeroFill     = 0x0000,
   iipmOcclusionFilling_UnitFill     = 0x0080,
   iipmOcclusionFilling_Mask         = 0x0080,

   iipmBoundaryExtraction_Undefined = 0x0000,
   iipmBoundaryExtraction_Processed = 0x0100,
   iipmBoundaryExtraction_Mask      = 0x0100,
};
```

6.4.28. eIrisImageFormat

eIrisImageFormat enumeration contains a set of constants specifying the format of images stored in a single record with iris graphic data of the document owner (see section 6.3.41).

```
enum eIrisImageFormat
{
   iifMono_Raw       = 0x0002,
   iifRGB_Raw        = 0x0004,
   iifMono_JPEG      = 0x0006,
   iifRGB_JPEG       = 0x0008,
   iifMono_JPEG_LS   = 0x000A,
   iifRGB_JPEG_LS    = 0x000C,
   iifMono_JPEG2000 = 0x000E,
   iifRGB_JPEG2000  = 0x0010,
};
```

Constants correspond to various formats of image records:

iifMono_Raw	–	grayscale, without compression;
iifRGB_Raw	–	RGB, without compression;
iifMono_JPEG	–	grayscale, JPEG compression;
iifRGB_JPEG	–	RGB, JPEG compression;
iifMono_JPEG_LS	–	grayscale, lossless JPEG compression;
iifRGB_JPEG_LS	–	RGB, lossless JPEG;
iifMono_JPEG2000	–	grayscale, JPEG-2000 compression;
iifRGB_JPEG2000	–	RGB, JPEG-2000 compression.

6.4.29. eIrisImageTransformation

eIrisImageTransformation enumeration contains a set of constants specifying the type of conversion to the polar coordinates for images stored in a single record of iris graphic data of the document owner (see section 6.3.41).

```
enum eIrisImageTransformation
{
```

```
   iitUndefined   = 0,
   iitStandard    = 1,
};
```

Value of constants:

`iitUndefined` – undefined;
`iitStandard` – standard conversion.

6.4.30. eIrisSubtype

`eIrisSubtype` enumeration contains a set of constants specifying the type of iris image template from the record of biometric graphic data (see section 6.3.42).

```
enum eIrisSubtype
{
   iftUndefined   = 0,
   iftEyeRight    = 1,
   iftEyeLeft     = 2,
};
```

Value of constants:

`iftUndefined` – undefined;
`iftEyeRight` – right eye;
`iftEyeLeft` – left eye.

6.4.31. eMinutiaeExtendedDataType

`eMinutiaeExtendedDataType` enumeration contains a set of constants specifying the type of additional information on the coded fingerprint (see section 6.3.34).

```
enum eMinutiaeExtendedDataType
{
   medtReserved          = 0x0000,
   medtRidgeCountData    = 0x0001,
   medtCoreAndDeltaData  = 0x0002,
   medtZonalQualityData  = 0x0003,
};
```

Value of constants:

`medtRidgeCountData` – pData field of **`TMinutiaeExtData`** structure contains a pointer to a structure **`TMinutiaeRidgeCountData`**;
`medtCoreAndDeltaData` – pData field of **`TMinutiaeExtData`** structure contains a pointer to a structure **`TCoreAndDeltaData`**;
`medtZonalQualityData` – pData field of **`TMinutiaeExtData`** structure contains a pointer to a structure **`TZonalQualityData`**.

6.4.32. eRidgeCountExtractionMethod

eRidgeCountExtractionMethod enumeration contains a set of constants specifying the ridge extraction method between the pairs of minutiae in the additional information on the encoded fingerprint (see section 6.3.35).

```
enum eRidgeCountExtractionMethod
{
   rcemNonSpecific    = 0x00,
   rcemFourNeighbor   = 0x01,
   rcemEightNeighbor  = 0x02,
};
```

Value of constants:

rcemNonSpecific	–	method undefined;
rcemFourNeighbor	–	by four neighboring areas;
rcemEightNeighbor	–	by eight neighboring areas.

6.4.33. CDocFormat

CDocFormat enumeration contains a set of constants specifying the document type by the classification of document formats from ISO/IEC 7810 (see section 6.3.20).

```
enum CDocFormat
{
   dfID1  = 0,
   dfID2  = 1,
   dfID3  = 2,
   dfNON  = 3,
};
```

Value of constants of document format:

dfID1	–	ID1;
dfID2	–	ID2;
dfID3	–	ID3;
dfNON	–	undefined.

Note. Here only the values are given, which are used in the process of RFID SDK work.

6.4.34. eRFID_VisualFieldType

eRFID_VisualFieldType enumeration contains a set of constants specifying the type of logically parsed fields of document filling (see sections 6.3.6, 6.3.15–6.3.19).

```
enum eRFID_VisualFieldType
{
   ft_SBH_SecurityOptions          = 300,
   ft_SBH_IntegrityOptions         = 301,
```

```
    ft_Date_of_Creation                 = 302,
    ft_Validity_Period                  = 303,
    ft_Patron_Header_Version            = 304,
    ft_BDB_Type                         = 305,
    ft_Biometric_Type                   = 306,
    ft_Biometric_Subtype                = 307,
    ft_Biometric_ProductID              = 308,
    ft_Biometric_Format_Owner           = 309,
    ft_Biometric_Format_Type            = 310,
    ft_Phone                            = 311,
    ft_Profession                       = 312,
    ft_Title                            = 313,
    ft_Personal_Summary                 = 314,
    ft_Other_Valid_ID                   = 315,
    ft_Custody_Info                     = 316,
    ft_Other_Name                       = 317,
    ft_Observations                     = 318,
    ft_Tax                              = 319,
    ft_Date_of_Personalization          = 320,
    ft_Personalization_SN               = 321,
    ft_Date_of_Record                   = 322,
    ft_PersonToNotify_Date_of_Record    = 323,
    ft_PersonToNotify_Name              = 324,
    ft_PersonToNotify_Phone             = 325,
    ft_PersonToNotify_Address           = 326,
    ft_DS_Certificate_Issuer            = 327,
    ft_DS_Certificate_Subject           = 328,
    ft_DS_Certificate_ValidFrom         = 329,
    ft_DS_Certificate_ValidTo           = 330,
    ft_VRC_DataObject_Entry             = 331,
};
```

Value of constants:

`ft_SBH_SecurityOptions`	–	parameters of biometric data protection ;
`ft_SBH_IntegrityOptions`	–	parameters of biometric data integrity;
`ft_Date_of_Creation`	–	date of creation of biometric data record;
`ft_Validity_Period`	–	term of validity of biometric data record;
`ft_Patron_Header_Version`	–	version of header of biometric data format owner;
`ft_BDB_Type`	–	type of biometric data record;
`ft_Biometric_Type`	–	type of biometric data;
`ft_Biometric_Subtype`	–	subtype of biometric data;
`ft_Biometric_ProductID`	–	identifier of biometric data;
`ft_Biometric_Format_Owner`	–	identifier of biometric data format owner;
`ft_Biometric_Format_Type`	–	biometric data format;
`ft_Phone`	–	DO’s phone number;
`ft_Profession`	–	DO’s profession;
`ft_Title`	–	DO’s title;
`ft_Personal_Summary`	–	DO’s personal summary data;
`ft_Other_Valid_ID`	–	other valid identifier;

`ft_Custody_Info` – custody information;
`ft_Other_Name` – other name;
`ft_Observations` – observations;
`ft_Tax` – tax information;
`ft_Date_of_Personalization` – date of document personalization;
`ft_Personalization_SN` – serial number of personalization;
`ft_Date_of_Record` – date of record entry;
`ft_PersonToNotify_Date_of_Record` – date of record entry on persons to notify in case of emergency;
`ft_PersonToNotify_Name` – name of person to notify in case of emergency;
`ft_PersonToNotify_Phone` – phone number of person to notify in case of emergency;
`ft_PersonToNotify_Address` – address of person to notify in case of emergency;
`ft_DS_Certificate_Issuer` – textual information about the DS-certificate issuer (see section 6.3.55);
`ft_DS_Certificate_Subject` – textual information about the document issuer (see section 6.3.55);
`ft_DS_Certificate_ValidFrom` – start date of the DS-certificate validity;
`ft_DS_Certificate_ValidTo` – expiration date of the DS-certificate,
`ft_VRC_DataObject_Entry` – vehicle category/restrictions/conditions from `DG1` data group of *`eDL`* application [39].

6.4.35. eVisualFieldType

`eVisualFieldType` enumeration contains a set of constants specifying the type of logically parsed fields of document filling (see section 6.3.6, 6.3.15–6.3.19).

```
enum eVisualFieldType
{
   ft_Document_Class_Code        = 0,
   ft_Issuing_State_Code         = 1,
   ft_Document_Number            = 2,
   ft_Date_of_Expiry             = 3,
   ft_Date_of_Issue              = 4,
   ft_Date_of_Birth              = 5,
   ft_Place_of_Birth             = 6,
   ft_Personal_Number            = 7,
   ft_Surname                    = 8,
   ft_Given_Names                = 9,
   ft_Nationality                = 11,
   ft_Sex                        = 12,
   ft_Address                    = 17,
   ft_Authority                  = 24,
   ft_Surname_And_Given_Names    = 25,
   ft_Nationality_Code           = 26,
   ft_Other                      = 50,
   ft_Address_State              = 65,
   ft_Address_Street             = 76,
```

```
   ft_Address_City              = 77,
   ft_Artistic_Name             = 254,
   ft_Academic_Title            = 255,
   ft_Address_Country           = 256,
   ft_Address_Zipcode           = 257,
   ft_eID_Residence_Permit1     = 258,
   ft_eID_Residence_Permit2     = 259,
   ft_eID_PlaceOfBirth_Street   = 260,
   ft_eID_PlaceOfBirth_City     = 261,
   ft_eID_PlaceOfBirth_State    = 262,
   ft_eID_PlaceOfBirth_Country  = 263,
   ft_eID_PlaceOfBirth_Zipcode  = 264,
};
```

Note. *Here only the values are given, which are used in the process of RFID SDK work.*

Value of constants:

`ft_Document_Class_Code`	–	document class code;
`ft_Issuing_State_Code`	–	issuing state code by Doc 9303 ICAO;
`ft_Document_Number`	–	document number;
`ft_Date_of_Expiry`	–	date of expiry of the document;
`ft_Date_of_Issue`	–	date of issue of the document;
`ft_Date_of_Birth`	–	DO's date of birth;
`ft_Place_of_Birth`	–	DO's place of birth;
`ft_Personal_Number`	–	personal number;
`ft_Surname`	–	surname;
`ft_Given_Names`	–	given names;
`ft_Nationality`	–	nationality;
`ft_Sex`	–	sex;
`ft_Address`	–	address;
`ft_Authority`	–	issuing authority;
`ft_Surname_And_Given_Names`	–	surname and given names;
`ft_Nationality_Code`	–	nationality code by Doc 9303 ICAO;
`ft_Other`	–	other information;
`ft_Artistic_Name`	–	artistic/religious name (alias);
`ft_Academic_Title`	–	academic title;
`ft_Address_Country`	–	address (country);
`ft_Address_Zipcode`	–	address (zip code);
`ft_Address_Street`	–	address (street);
`ft_Address_City`	–	address (city);
`ft_eID_Residence_Permit1`	–	data on permanent residence permit (see section 4.3.2);
`ft_eID_Residence_Permit2`	–	data on permanent residence permit (see section 4.3.2);
`ft_eID_PlaceOfBirth_Street`	–	place of birth (street);
`ft_eID_PlaceOfBirth_City`	–	place of birth (city);
`ft_eID_PlaceOfBirth_State`	–	place of birth (region);
`ft_eID_PlaceOfBirth_Country`	–	place of birth (country);

`ft_eID_PlaceOfBirth_Zipcode` – place of birth (zip code).

6.4.36. eGraphicFieldType

eGraphicFieldType enumeration contains a set of constants specifying the logical type of graphic fields of biometric data record (see section 6.3.8).

```
enum eGraphicFieldType
{
   gf_Portrait               = 201,
   gf_Fingerprint            = 202,
   gf_Eye                    = 203,
   gf_Signature              = 204,
   gt_BarCode                = 205,
   gt_Proof_Of_Citizenship = 206,
   gt_Document_Front         = 207,
   gt_Document_Rear          = 208,
   gt_Other                  = 250,
};
```

Logical types of fields defined by the constants of this enumeration:

`gf_Portrait`	–	DO's photo;
`gf_Fingerprint`	–	DO's fingerprint;
`gf_Eye`	–	DO's iris image;
`gf_Signature`	–	DO's signature;
`gt_BarCode`	–	barcode image;
`gt_Proof_Of_Citizenship`	–	image of document proving DO's citizenship;
`gt_Document_Front`	–	image of document face;
`gt_Document_Rear`	–	image of document rear side;
`gt_Other`	–	undefined type of image.

6.4.37. eMIFARE_KeyMode

eMIFARE_KeyMode enumeration contains a set of constants specifying the mode of authentication when reading data from the RFID-chip via MIFARE® Classic Protocol (see section 5.7.3).

```
enum eMIFARE_KeyMode
{
   mkmDefault      = 1,
   mkmSingleKey    = 2,
   mkmFullKeyTable = 3,
};
```

Value of constants of authentication mode:

`mkmDefault`	–	default;
`mkmSingleKey`	–	one key use for all memory sectors;

`mkmFullKeyTable` – individual key use for all memory sectors.

6.4.38. eOutputFormat

`eOutputFormat` enumeration contains a set of constants specifying the format of return data when receiving results by **`_RFID_CheckResult()`** function (see sections 5.6.3, 6.1.5).

```
enum eOutputFormat
{
   ofDefault         = 0,
   ofClipboard_XML   = 3,
   ofFile_XML        = 4,
   ofXML             = 5,
};
```

Value of constants:

`ofDefault`	–	default mode. Only a pointer to the result data structure will be returned;
`ofClipboard_XML`	–	formation of result data structure XML-representation and its transfer through the clipboard;
`ofFile_XML`	–	formation of result data structure XML-representation and its recording to a file;
`ofXML`	–	formation of result data structure XML-representation.

6.4.39. eOutputFormatField

`eOutputFormatField` enumeration contains a set of constants specifying the mechanism of data transfer when using **`_RFID_CheckResultFromList()`** function (see sections 5.6.3, 6.1.6).

```
enum eOutputFormatField
{
   offInfo        = 0,
   offClipboard   = ofrTransport_Clipboard,
   offFile        = ofrTransport_File,
   offXML         = ofrFormat_XML,
   offFileBuffer  = ofrFormat_FileBuffer,
};
```

Value of constants:

`offClipboard`	–	transfer of text and graphic field contents through the clipboard;
`offFile`	–	recording the graphic field contents of in the file;
`offXML`	–	formation of XML-representation of the result;
`offFileBuffer`	–	request of the image of the graphic file.

6.4.40. eRFID_ResultStatus

eRFID_ResultStatus enumeration contains a set of constants being the return codes from **_RFID_CheckResult()** and **_RFID_CheckResultFromList()** functions (see sections 5.6.3, 6.1.5, 6.1.6).

```
enum eRFID_ResultStatus
{
   RFID_ResultStatus_NotAvailable    = 0xffffffff,
   RFID_ResultStatus_EndOfList       = 0xfffffffe,
   RFID_ResultStatus_InvalidParameter= 0xfffffffd,
   RFID_ResultStatus_Error           = 0xfffffffc,
};
```

Value of constants:

RFID_ResultStatus_NotAvailable	–	requested type of the result is not available;
RFID_ResultStatus_EndOfList	–	the end of the list was reached during the previous step of receiving result data, and there are no new data in the processed list;
RFID_ResultStatus_InvalidParameter	–	invalid parameter of function call;
RFID_ResultStatus_Error	–	error in formation of additional result representation (when saving a file, placing in the clipboard or converting to XML format).

6.4.41. eRFID_NotificationCodes

eRFID_NotificationCodes enumeration contains a set of notification codes transferred to the user application by calling the callback-function (see section 6.2).

```
enum eRFID_NotificationCodes
{
   RFID_Notification_Error                     = 0x00010000,
   RFID_Notification_DocumentReady             = 0x00010001,
   RFID_Notification_ReadProtocol4             = 0x00010003,
   RFID_Notification_ReadProtocol3             = 0x0001000A,
   RFID_Notification_Progress                  = 0x0001000B,
   RFID_Notification_PA_Request                = 0x00013000,
   RFID_Notification_TA_Step                   = 0x0001000E,
   RFID_Notification_SM_Required               = 0x0001000F,
   RFID_Notification_SM_Established            = 0x0001400F,
   RFID_Notification_ISOError                  = 0x00011000,
   RFID_Notification_PCSC_ReaderDisconnected   = 0x00020000,
   RFID_Notification_PCSC_ReaderListChanged    = 0x00020001,
   RFID_Notification_PCSC_ReaderListChanging   = 0x00020008,
   RFID_Notification_PCSC_BytesReceived        = 0x00020002,
   RFID_Notification_PCSC_TotalReadingTime     = 0x00020003,
   RFID_Notification_PCSC_DataReceived         = 0x00020004,
```

```
   RFID_Notification_PCSC_BytesSent           = 0x00020005,
   RFID_Notification_PCSC_TotalReadingSpeed   = 0x00020006,
   RFID_Notification_PCSC_TotalProcessTime    = 0x00020007,
   RFID_Notification_PCSC_ExtLengthSupport    = 0x00020010,
   RFID_Notification_PA_CertificateChain      = 0x00020011,
   RFID_Notification_PA_CertificateChainItem  = 0x00020012,
   RFID_Notification_Scenario                 = 0x00020020,

   //composite notification codes
   RFID_Notification_PCSC_ReadingDatagroup    = 0x00030000,
   RFID_Notification_PCSC_FileNotFound        = 0x00040000,
   RFID_Notification_PCSC_EndOfFile           = 0x00050000,
   RFID_Notification_PCSC_FileAccessDenied    = 0x00060000,
   RFID_Notification_PCSC_ApplicationSelected = 0x00070000,
   RFID_Notification_ACProcedure_Start        = 0x00080000,
   RFID_Notification_ACProcedure_Finish       = 0x00090000,
   RFID_Notification_PA_SecurityObjectCheck   = 0x000A0000,
   RFID_Notification_PA_FileCheck             = 0x000B0000,
   RFID_Notification_PCSC_UpdatingDatagroup   = 0x000C0000,
   RFID_Notification_AuxiliaryDataValidation  = 0x000D0000,
   RFID_Notification_RI_SectorID              = 0x000E0000,
   RFID_Notification_Biometrics_EmptyPlaceholder = 0x000F0000,
};
```

Value of notification codes:

- `RFID_Notification_Error`

Error, `value` contains an error code from **`eRFID_ErrorCodes`**

- `RFID_Notification_DocumentReady`

Event of appearance of RFID-chip in the scope of the reader antenna or its moving away from the scope of the reader. Parameter `value` contains a flag of presence of RFID-chip in the scope of the reader (`true` or `false`)

- `RFID_Notification_ReadProtocol4`

event of the beginning/end of data reading from the RFID-chip via ISO/IEC 14443-4 protocol When working in the batch mode. Parameter `value` contains `false` at the beginning of reading and `true` at the end

- `RFID_Notification_ReadProtocol3`

Event of the beginning/end of data reading from the RFID-chip via ISO/IEC 14443-3 protocol. Parameter `value` contains `false` at the beginning of reading and `true` at the end

- `RFID_Notification_Progress`

Indication of the progress of execution of data reading operation (see sections 5.7.4, 5.8.10)

- `RFID_Notification_PA_Request`

Request of the user-defined DS-certificate priot to the procedure of digital signature verification of `EF.SOD` document security object in the batch mode (see section 5.7.7)

- `RFID_Notification_TA_Step`

Indication of the next step of terminal authentication in *Online-authentication* mode (see section 5.8.15)

- `RFID_Notification_SM_Required`

Event of detection of the need to organize a secure communication channel (see sections 5.7.5, 5.8.8)

- `RFID_Notification_SM_Established`

Event of the result of the opening of a secure communication channel (see section 5.7.5, 5.8.8)

- `RFID_Notification_ISOError`

Event informing the user application on detection of data incompliance processed with the regulations of normative documents, errors when executing the current operation. `value` parameter contains error code (one of **`eLDS_ParsingErrorCodes`** or **`eLDS_ParsingNotificationCodes`** values)

- `RFID_Notification_PCSC_ReaderDisconnected`

Event of unplugging of the RFID-chip reader from the PC

- `RFID_Notification_PCSC_ReaderListChanging`

Event of the beginning of reorganization of the list of RFID-readers connected to the PC, working under PC/SC-driver control (see section 5.3)

- `RFID_Notification_PCSC_ReaderListChanged`

Event of the end of reorganization of the list of RFID-readers connected to the PC, working under PC/SC-driver control (see section 5.3)

- `RFID_Notification_PCSC_BytesReceived`

Transfer of the total amount of information received from the RFID-chip to the user application during execution of data reading operation (see section 5.7.8)

- `RFID_Notification_PCSC_TotalReadingTime`

Transfer of the total time of execution of data reading operation to the user application (see section 5.7.8)

- `RFID_Notification_PCSC_DataReceived`

Transfer of the total amount of information and service groups data received from the RFID-chip to the user application during execution of data reading operation (see section 5.7.8)

- `RFID_Notification_PCSC_BytesSent`

Transfer of the total amount of information transmitted to the RFID-chip to the user application during execution of data reading operation (see section 5.7.8)

- RFID_Notification_PCSC_TotalReadingSpeed

Transfer of the average data reading rate to the user application (see section 5.7.8)

- RFID_Notification_PCSC_TotalProcessTime

Transfer of the total time of execution of data reading procedure to the user application (see section 5.7.8)

- RFID_Notification_PCSC_ExtLengthSupport

Event of detection of extended length reading commands support by the RFID-chip (see section 5.4.4)

- RFID_Notification_PA_CertificateChain

Event of the start/end of the certificate chain formation for the document security object digital signature verification as a part of passive authentication procedure (see sections 5.7.7, 5.8.12). Parameter `false` – beginning of the operation, `true` – end.

- RFID_Notification_PA_CertificateChainItem

Event that indicates a type of the current analyzed element of the certificate chain being composed (see section 5.8.12). `value` contains one of **eRFID_CertificateType** codes. All subsequent notifications prior to the next `RFID_Notification_PA_CertificateChainItem` or `RFID_Notification_PA_CertificateChain` will correspond to this element.

- RFID_Notification_Scenario

A request from the user application of some data or actions in a certain step of the scenario (see section 5.9.3). As a parameter acts `VARIANT *` pointer to XML-string defining a concrete step of the scenario, which is also the receiver of data requested.

- RFID_Notification_PCSC_ReadingDatagroup

Event of the beginning/end of file reading. The low order `WORD` contains a file identifier from **eRFID_DataFile_Type** (see sections 5.7.4, 5.8.10)

- RFID_Notification_PCSC_FileNotFound

Event of detection of file absence. The low order `WORD` contains a file identifier from **eRFID_DataFile_Type** (see sections 5.7.4, 5.8.10)

- RFID_Notification_PCSC_EndOfFile

Event of reaching the file end when performing its reading. The low order `WORD` contains a file identifier from **eRFID_DataFile_Type** (see section 5.7.4, 5.8.10)

- RFID_Notification_PCSC_FileAccessDenied

Event of detection of absence of the file access rights. The low order `WORD` contains a file identifier from **eRFID_DataFile_Type** (see sections 5.7.4, 5.8.10)

- RFID_Notification_PCSC_ApplicationSelected

Event of the application selection operation. The low order `WORD` contains a file identifier from **eRFID_Application_Type**, `value` parameter – operation result (see section 5.8.9)

- RFID_Notification_ACProcedure_Start

Event of the beginning of the authentication or secure data access procedure. The low order WORD contains a procedure identifier from **eRFID_AccessControl_ProcedureType** (see sections 5.7.6, 5.8.7)

- RFID_Notification_ACProcedure_Finish

Event of the end of the authentication or secure data access procedure.. The low order WORD contains a procedure identifier from **eRFID_AccessControl_ProcedureType**, value parameter – operation result (see sections 5.7.6, 5.8.7)

- RFID_Notification_PA_SecurityObjectCheck

Event of the data security object verification as part of PA. The low order WORD contains an identifier of the file, which is a source of the security object (value from **eRFID_DataFile_Type**), value parameter – operation result (see sections 5.7.7, 5.8.12)

- RFID_Notification_PA_FileCheck

Event of the file data integrity checking as part of PA. The low order WORD contains a file identifier (value from **eRFID_DataFile_Type**), value parameter – operation result (see sections 5.7.7, 5.8.13)

- RFID_Notification_PCSC_UpdatingDatagroup

Event of the procedure of file contents updating. The low order WORD contains a file identifier from **eRFID_DataFile_Type** (see section 5.8.19)

- RFID_Notification_AuxiliaryDataValidation

Event of the auxiliary data verification. The low order WORD contains a type of the verified data (value from **eRFID_AuxiliaryDataType**), value parameter – operation result (see section 5.8.18)

- RFID_Notification_RI_SectorID

Event of the receiving of the sector identifier data during RI. The low order WORD contains a type identifier of the sector key (value from **eRFID_SectorKeyType**), value parameter – a pointer to the corresponding identifier data container (see section 5.8.17)

- RFID_Notification_Biometrics_EmptyPlaceholder

Event of the detection of real biometric data absence in DG3 or DG4 and random filling data usage [35, R7-p1_v2_sIII_0057, R7-p3_v2_sIII_0011]. The low order WORD contains a file identifier from **eRFID_DataFile_Type**.

6.4.42. eLDS_ParsingErrorCodes

eLDS_ParsingErrorCodes enumeration contains a set of critical remarks detected during analysis of data structure used during SDK work (see section 5.2).

```
enum eLDS_ParsingErrorCodes
{
   errLDS_Ok                                          = 0x00000001,
   errLDS_ASN_IncorrectData                           = 0x80000001,
   errLDS_ASN_NotEnoughData                           = 0x80000002,
   errLDS_ASN_Contents_UnexpectedData                 = 0x80000003,

   errLDS_ASN_SignedData_IncorrectData                = 0x80000008,
   errLDS_ASN_SignedData_EncapContents_IncorrectData  = 0x80000009,
   errLDS_ASN_SignedData_Version_IncorrectData        = 0x8000000A,
   errLDS_ASN_SignedData_DigestAlgorithms_IncorrectData = 0x80000011,

   errLDS_ASN_LDSObject_IncorrectData                 = 0x80000013,
   errLDS_ASN_LDSObject_Version_IncorrectData         = 0x80000014,
   errLDS_ASN_LDSObject_DigestAlgorithm_IncorrectData = 0x80000015,
   errLDS_ASN_LDSObject_DGHashes_IncorrectData        = 0x80000016,
   errLDS_ASN_LDSObject_VersionInfo_IncorrectData     = 0x80000012,

   errLDS_ASN_Certificate_IncorrectData               = 0x80000017,
   errLDS_ASN_Certificate_Version_IncorrectData       = 0x80000018,
   errLDS_ASN_Certificate_SN_IncorrectData            = 0x80000019,
   errLDS_ASN_Certificate_Signature_IncorrectData     = 0x8000001A,
   errLDS_ASN_Certificate_Issuer_IncorrectData        = 0x8000001B,
   errLDS_ASN_Certificate_Validity_IncorrectData      = 0x8000001C,
   errLDS_ASN_Certificate_Subject_IncorrectData       = 0x8000001D,
   errLDS_ASN_Certificate_SubjectPK_IncorrectData     = 0x8000001E,
   errLDS_ASN_Certificate_Extensions_IncorrectData    = 0x8000001F,

   errLDS_ASN_SignerInfo_IncorrectData                = 0x80000020,
   errLDS_ASN_SignerInfo_Version_IncorrectData        = 0x80000021,
   errLDS_ASN_SignerInfo_SID_IncorrectData            = 0x80000022,
   errLDS_ASN_SignerInfo_DigestAlg_IncorrectData      = 0x80000023,
   errLDS_ASN_SignerInfo_SignedAttrs_IncorrectData    = 0x80000024,
   errLDS_ASN_SignerInfo_SignAlg_IncorrectData        = 0x80000025,
   errLDS_ASN_SignerInfo_Signature_IncorrectData      = 0x80000026,
   errLDS_ASN_SignerInfo_UnsignedAttrs_IncorrectData  = 0x80000027,

   errLDS_ICAO_LDSObject_UnsupportedDigestAlgorithm   = 0x80000030,
   errLDS_ICAO_SignedData_SignerInfos_Empty           = 0x80000031,
   errLDS_ICAO_SignerInfo_UnsupportedDigestAlgorithm  = 0x80000032,
   errLDS_ICAO_SignerInfo_UnsupportedSignatureAlgorithm = 0x80000033,
   errLDS_ICAO_SignerInfo_MessageDigestError          = 0x80000034,
   errLDS_ICAO_SignerInfo_SignedAttrs_Missed          = 0x80000036,

   errLDS_Auth_SignerInfo_CantFindCertificate         = 0x80000035,

   errLDS_Auth_Error                                  = 0x80000050,
   errLDS_Auth_UnsupportedSignatureAlgorithm          = 0x80000051,
   errLDS_Auth_UnsupportedPublicKeyAlgorithm          = 0x80000052,
   errLDS_Auth_MessedAlgorithms                       = 0x80000053,
   errLDS_Auth_PublicKeyDataInvalid                   = 0x80000054,
   errLDS_Auth_AlgorithmParametersDataInvalid         = 0x80000055,
```

```
errLDS_Auth_SignatureDataInvalid                        = 0x80000056,
errLDS_Auth_UnsupportedDigestAlgorithm                  = 0x80000057,
errLDS_Auth_SignatureDataIncorrect                      = 0x80000058,
errLDS_Auth_AlgorithmParametersNotDefined               = 0x80000059,
errLDS_Auth_SignatureCheckFailed                        = 0x8000005A,

errLDS_DG_WrongTag                                      = 0x80000070,
errLDS_DG_Contents_UnexpectedData                       = 0x80000071,

errLDS_BAP_SymmetricCypher_CantInitialize               = 0x81000011,

errLDS_PACE_Info_NotAvailable                           = 0x81000020,
errLDS_PACE_SymmetricCypher_CantInitialize              = 0x81000021,
errLDS_PACE_KeyAgreement_CantInitialize                 = 0x81000022,
errLDS_PACE_EphemeralKeys_CantCreate                    = 0x81000023,
errLDS_PACE_Mapping_CantDecodeNonce                     = 0x81000024,
errLDS_PACE_SharedSecret_CantCreate                     = 0x81000025,
errLDS_PACE_DomainParams_UnsupportedFormat              = 0x81000026,
errLDS_PACE_EphemeralKeys_Incorrect                     = 0x81000027,
errLDS_PACE_Mapping_EphemeralKeys_Incorrect             = 0x81000028,
errLDS_PACE_Mapping_CantPerform                         = 0x81000029,
errLDS_PACE_NonMatchingAuthTokens                       = 0x8100002A,

errLDS_PACE_CAM_Data_Incorrect                          = 0x8100002B,
errLDS_PACE_CAM_Data_CantVerify                         = 0x8100002C,
errLDS_PACE_CAM_Data_NonMatching                        = 0x8100002D,

errLDS_PACE_IM_Scheme_Incorrect                         = 0x8100002E,
errLDS_PACE_IM_RandomMapping_Failed                     = 0x8100002F,

errLDS_CA_CantFindPublicKey                             = 0x81000030,
errLDS_CA_CantFindInfo                                  = 0x81000031,
errLDS_CA_IncorrectVersion                              = 0x81000032,
errLDS_CA_CantFindDomainParameters                      = 0x81000033,
errLDS_CA_KeyAgreement_CantInitialize                   = 0x81000034,
errLDS_CA_PublicKey_UnsupportedAlgorithm                = 0x81000035,
errLDS_CA_EphemeralKeys_CantCreate                      = 0x81000036,
errLDS_CA_SharedSecret_CantCreate                       = 0x81000037,
errLDS_CA_NonMatchingAuthTokens                         = 0x81000038,

errLDS_TA_IncorrectVersion                              = 0x81000040,
errLDS_TA_CantBuildCertificateChain                     = 0x81000041,
errLDS_TA_CantFindISPrivateKey                          = 0x81000042,
errLDS_TA_PublicKey_UnsupportedAlgorithm                = 0x81000043,
errLDS_TA_SignatureBuildingError                        = 0x81000044,
errLDS_TA_InvalidKeyAlgorithmParameters                 = 0x81000045,

errLDS_AA_PublicKey_UnsupportedAlgorithm                = 0x81000050,
errLDS_AA_PublicKey_IncorrectData                       = 0x81000051,
errLDS_AA_PublicKey_IncorrectParameters                 = 0x81000052,
errLDS_AA_PublicKey_UndefinedParameters                 = 0x81000053,
errLDS_AA_Signature_IncorrectData                       = 0x81000054,
errLDS_AA_UnsupportedRecoveryScheme                     = 0x81000055,
errLDS_AA_IncorrectTrailer                              = 0x81000056,
errLDS_AA_UnsupportedDigestAlgorithm                    = 0x81000057,

errLDS_RI_SectorKey_CantFind                            = 0x81000070,
errLDS_RI_SectorKey_IncorrectData                       = 0x81000071,
```

```
   errLDS_RI_SectorKey_IncompleteData                     = 0x81000072,

   errLDS_CV_Certificate_MissingMandatoryData_PK          = 0x81000060,
   errLDS_CV_Certificate_PublicKey_Unsupported            = 0x81000062,
   errLDS_CV_Certificate_CHAT_UnsupportedTerminalType     = 0x81000063,
   errLDS_CV_Certificate_PrivateKey_Unsupported           = 0x81000064,
   errLDS_CV_Certificate_PrivateKey_InvalidParams         = 0x81000065,
   errLDS_CV_Certificate_IncorrectData                    = 0x81000160,
   errLDS_CV_Certificate_CPI_IncorrectData                = 0x81000161,
   errLDS_CV_Certificate_CAR_IncorrectData                = 0x81000162,
   errLDS_CV_Certificate_PublicKey_IncorrectData          = 0x81000163,
   errLDS_CV_Certificate_CHR_IncorrectData                = 0x81000164,
   errLDS_CV_Certificate_CHAT_IncorrectData               = 0x81000165,
   errLDS_CV_Certificate_ValidFrom_IncorrectData          = 0x81000166,
   errLDS_CV_Certificate_ValidTo_IncorrectData            = 0x81000167,
   errLDS_CV_Certificate_Extensions_IncorrectData         = 0x81000168,
   errLDS_CV_Certificate_PrivateKey_IncorrectData         = 0x81000169,
   errLDS_CV_Certificate_PrivateKey_Missing               = 0x8100016A,
};
```

Constants describe appearance of the following situations:

- `errLDS_Ok`

No remarks.

- `errLDS_ASN_IncorrectData`

Provided ASN.1-data are incorrect (common case – impossible to form elementary ASN.1-objects).

- `errLDS_ASN_NotEnoughData`

Provided ASN.1-data are incorrect (not enough data).

- `errLDS_ASN_Contents_UnexpectedData`

Other contents expected for complex ASN.1-object components (by the structure, number of elements).

- `errLDS_ASN_SignedData_IncorrectData`

Incorrect format of `SignedData` ASN.1-object [7, §5.1], which is the content of the document security object [2, §A3.1], [24, part 3, §A.1.2.5] – common case.

- `errLDS_ASN_SignedData_EncapContents_IncorrectData`

Incorrect format of encapsulated `encapContentInfo` data of `SignedData` ASN.1-object [7, §5.1], which is the content of the document security object [2, §A3.1], [24, part 3, §A.1.2.5].

- `errLDS_ASN_SignedData_Version_IncorrectData`

Invalid `version` field format of `SignedData` ASN.1-object [7, §5.1], which is the content of the document security object [2, §A3.1], [24, part 3, §A.1.2.5].

- `errLDS_ASN_SignedData_DigestAlgorithms_IncorrectData`

Invalid format of `digestAlgorithms` field of `SignedData` ASN.1-object [7, §5.1], which is the content of the document security object [2, § A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_LDSObject_IncorrectData

Incorrect format of LDSSecurityObject ASN.1-object, encapsulated in EF.SOD document security object (see section 4.3.1) [2, §A3.2].

- errLDS_ASN_LDSObject_Version_IncorrectData

Incorrect format of version field of LDSSecurityObject ASN.1-object, encapsulated in EF.SOD document security object (see section 4.3.1) [2, § A3.2].

- errLDS_ASN_LDSObject_DigestAlgorithm_IncorrectData

Incorrect format of hashAlgorithm field of LDSSecurityObject ASN.1-object, encapsulated in EF.SOD document security object (see section 4.3.1) [2, §A3.2].

- errLDS_ASN_LDSObject_DGHashes_IncorrectData

Incorrect format of dataGroupHashes field of LDSSecurityObject ASN.1-object, encapsulated in EF.SOD document security object (see section 4.3.1) [2, §A3.2].

- errLDS_ASN_LDSObject_VersionInfo_IncorrectData

Incorrect format of ldsVersionInfo field of LDSSecurityObject ASN.1-object, encapsulated in EF.SOD document security object (for LDS version 1.8) [31, §2.2].

- errLDS_ASN_Certificate_IncorrectData

Incorrect format of Certificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_Version_IncorrectData

Incorrect format of version field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_SN_IncorrectData

Incorrect format of serialNumber field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_Signature_IncorrectData

Incorrect format of signature field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_Issuer_IncorrectData

Incorrect format of issuer field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_Validity_IncorrectData

Incorrect format of validity field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_Subject_IncorrectData

Incorrect format of subject field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_SubjectPK_IncorrectData

Incorrect format of subjectPublicKeyInfo field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_Certificate_Extensions_IncorrectData

Incorrect format of extensions field of TBSCertificate ASN.1-object [6, §4.1].

- errLDS_ASN_SignerInfo_IncorrectData

Incorrect format of signerInfos field of SignedData ASN.1-object [7, §5.1], which is the contents of the document security object [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_SignerInfo_Version_IncorrectData

Incorrect format of version field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_SignerInfo_SID_IncorrectData

Incorrect format of sid field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_SignerInfo_DigestAlg_IncorrectData

Incorrect format of digestAlgorithm field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_SignerInfo_SignedAttrs_IncorrectData

Incorrect format of signedAttrs field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, section IV, §A3.1].

- errLDS_ASN_SignerInfo_SignAlg_IncorrectData

Incorrect format of signatureAlgorithm field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, section IV, § A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_SignerInfo_Signature_IncorrectData

Incorrect format of signature field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- errLDS_ASN_SignerInfo_UnsignedAttrs_IncorrectData

Incorrect format of unsignedAttrs field of SignedInfo ASN.1-object [7, §5.3], containing data of digital signature of the document security object [2, секция IV, §A3.1].

- errLDS_ICAO_LDSObject_UnsupportedDigestAlgorithm

hashAlgorithm field of LDSSecurityObject ASN.1-object, encapsulated in EF.SOD document security object (see section 4.3.1), contains identifier (OID) of the unsupported hash algorithm [2, §8.5, section IV, §A3.2].

- errLDS_ICAO_SignedData_SignerInfos_Empty

No digital signature data object was found in the structure of the document security object (signerInfos list of SignedData ASN.1-object [7, §5.1] is empty).

- errLDS_ICAO_SignerInfo_UnsupportedDigestAlgorithm

digestAlgorithm field of SignerInfo ASN.1-object of document security object digital signature [7, § 5.3] contains identifier (OID) of unsupported hash algorithm [2, section IV, §8.5, §A3.2], [24, §A.1].

- `errLDS_ICAO_SignerInfo_UnsupportedSignatureAlgorithm`

`signatureAlgorithm` field of `SignerInfo` ASN.1-object of document security object digital signature [7, §5.3] contains identifier (OID) of unsupported digital signature algorithm [2, section IV, §8, §A3.2], [24, §A.1].

- `errLDS_ICAO_SignerInfo_MessageDigestError`

Error the of message digest calculation.

- `errLDS_ICAO_SignerInfo_SignedAttrs_Missed`

Mandatory `signedAttrs` field of `SignerInfo` ASN.1-object of `EF.SOD` document security object is not found [7, §5.3], [2, секция IV, §A3.1].

- `errLDS_Auth_SignerInfo_CantFindCertificate`

DS-certificate to verify the digital signature is not found.

- `errLDS_Auth_Error`

Digital signature verification failed (common case).

- `errLDS_Auth_UnsupportedSignatureAlgorithm`

Unsupported digital signature algorithm.

- `errLDS_Auth_UnsupportedPublicKeyAlgorithm`

Unsupported digital signature public key algorithm.

- `errLDS_Auth_MessedAlgorithms`

Digital signature algorithm does not correspond to the public key algorithm.

- `errLDS_Auth_PublicKeyDataInvalid`

Incorrect format of the public key data [5, §2.3].

- `errLDS_Auth_AlgorithmParametersDataInvalid`

Incorrect format of the public key parameters data [5, §2.3].

- `errLDS_Auth_SignatureDataInvalid`

Incorrect format of the digital signature data [5, §2.2].

- `errLDS_Auth_SignatureDataIncorrect`

Incorrect data of the digital signature (by length or compliance with a range of valid values).

- `errLDS_Auth_UnsupportedDigestAlgorithm`

Unsupported data hash algorithm for the digital signature verification.

- `errLDS_Auth_AlgorithmParametersNotDefined`

ECDSA public key parameters are not defined. According to [5, §2.2] they are defined by `EcpkParameters` ASN.1-object by one of three possible ways: explicitly, by the named curve identifier, implicitly. In case when the parameters are assigned implicity or the specified identifier of the curve is not supported, a procedure of verifying the digital signature becomes impossible.

- `errLDS_Auth_SignatureCheckFailed`

Digital signature verification failed.

- `errLDS_DG_WrongTag`

Incorrect file ASN.1-data tag – the actual value is not as expected (see section 6.4.3).

- `errLDS_DG_Contents_UnexpectedData`

Incorrect format of the file ASN.1-data [2, Section III, Appendix A], [24, part 2, A].

- `errLDS_BAP_SymmetricCypher_CantInitialize`

Can't create specified cypher object for BAP procedure [38, annex B].

- `errLDS_PACE_Info_NotAvailable`

PACE parameters are not defined at the time of the procedure – the corresponding `PACEInfo` is not found [23, §5.3.1]

- `errLDS_PACE_SymmetricCypher_CantInitialize`

Error of symmetric cipher object creating for PACE [23, §3.4]

- `errLDS_PACE_KeyAgreement_CantInitialize`

Error of key agreement object creating for PACE [23, §3.4]

- `errLDS_PACE_EphemeralKeys_CantCreate`

Error of creating a pair of ephemeral keys for PACE [23, §3.2]

- `errLDS_PACE_Mapping_CantDecodeNonce`

Error of nonce decoding, received from the chip, for use in the *mapping* operation [23, §3.4]

- `errLDS_PACE_SharedSecret_CantCreate`

Error of computing *shared secret* for PACE procedure[23, §3.2]

- `errLDS_PACE_DomainParams_UnsupportedFormat`

According to [23, §3.4.1.2] for the parameters of ECDH public key it is allowed to use only prime-curves with unpacked points [5, §2.3.5]

- `errLDS_PACE_EphemeralKeys_Incorrect`

PACE ephemeral public keys (terminal and chip) differ in the length or the same [23, §3.2]

- `errLDS_PACE_Mapping_EphemeralKeys_Incorrect`

PACE ephemeral public keys (terminal and chip) for use in the *mapping* operation differ in the length [23, §3.2]

- `errLDS_PACE_Mapping_CantPerform`

Error of PACE *mapping* operation – failed to compute the new key pair [23, §3.2]

- errLDS_PACE_NonMatchingAuthTokens

PACE authentication tokens of terminal and chip are different [23, §3.2]

- errLDS_PACE_CAM_Data_Incorrect

Incorrect PACE-CAM data received from the chip [23, §3.4.4, §3.5.3].

- errLDS_PACE_CAM_Data_CantVerify

Can't verify PACE-CAM data [23, §3.4.4].

- errLDS_PACE_CAM_Data_NonMatching

PACE-CAM data verification failed (length, contents) [23, §3.4.4].

- errLDS_PACE_IM_Scheme_Incorrect

Incorrect Integrated Mapping scheme (allowed Elliptic Curves only) [23, §3.4.2].

- errLDS_PACE_IM_RandomMapping_Failed

Can't perform Pseudo Random Mapping [23, §3.4.2.2.3].

- errLDS_CA_CantFindPublicKey

CA public key parameters are not defined at the time of the procedure – the corresponding ChipAuthenticationPublicKeyInfo is not found [1, §A.1.1.1], [24, part 3, §A.1.1.2]

- errLDS_CA_CantFindInfo

CA parameters (for version 2) are not defined at the time of the procedure – the corresponding ChipAuthenticationInfo is not found [24, part 3, §A.1.1.2]

- errLDS_CA_IncorrectVersion

Unsupported CA version in ChipAuthenticationInfo field of the ASN.1-object. The values 1 or 2 are allowed [1, §A.1.1.1], [24, part 3, §A.1.1.2]

- errLDS_CA_CantFindDomainParameters

Impossible to define the CA public key parameters at the time of the procedure;

- errLDS_CA_KeyAgreement_CantInitialize

Error of key object agreement creation for CA [24, part 1 §3.4, part 2 §3.3]

- errLDS_CA_PublicKey_UnsupportedAlgorithm

Unsupported CA public key algorithm [24, part 3, §A.4]

- errLDS_CA_EphemeralKeys_CantCreate

Error of creating a pair of ephemeral keys for CA [24, part 1 §3.4, part 2 §3.3]

- errLDS_CA_SharedSecret_CantCreate

Error of computing *shared secret* for CA [24, part 1 §3.4, part 2 §3.3]

- errLDS_CA_NonMatchingAuthTokens

CA authentication tokens of terminal and chip are different [24, part 1 §3.4, part 2 §3.3]

- `errLDS_TA_IncorrectVersion`

Unsupported TA version in `TerminalAuthenticationInfo` field of the ASN.1-object. The values `1` or `2` are allowed [1, §A.1.1.2], [24, part 3, §A.1.1.3]

- `errLDS_TA_CantBuildCertificateChain`

Error of building a certificate chain (see sections 4.9.3, 5.8.15). DV- and terminal certificate must be present at least

- `errLDS_TA_CantFindISPrivateKey`

Data of private key corresponding to the terminal certificate not found (see sections 4.9.3, 5.8.15)

- `errLDS_TA_PublicKey_UnsupportedAlgorithm`

Unsupported TA public key algorithm [1, §A.3], [24, part 3, §A.6]

- `errLDS_TA_SignatureBuildingError`

Error of TA digital signature calculation (see sections 4.9.3, 5.8.15)

- `errLDS_TA_InvalidKeyAlgorithmParameters`

Incorrect TA public key parameters [1, §C.3], [24, §D.3]

- `errLDS_AA_PublicKey_UnsupportedAlgorithm`

Unsupported AA public key algorithm [2, §A4]

- `errLDS_AA_PublicKey_IncorrectData`

Incorrect format of AA public key data [2, §A4.1]

- `errLDS_AA_PublicKey_IncorrectParameters`

Incorrect format of AA public key parameters data [2, §A4.1]

- `errLDS_AA_PublicKey_UndefinedParameters`

ECDSA public key parameters are not defined. According to [5, §2.2] they are defined by `EcpkParameters` ASN.1-object by one of three possible ways: explicitly, by the named curve identifier, implicitly. In case when the parameters are assigned implicity or the specified identifier of the curve is not supported, a procedure of AA becomes impossible

- `errLDS_AA_Signature_IncorrectData`

Incorrect data of AA digital signature (by length or compliance with a range of valid values)

- `errLDS_AA_UnsupportedRecoveryScheme`

Unsupported message recovery scheme. According to [2, §A4.2] only algorithm of partial recovery is supported

- `errLDS_AA_IncorrectTrailer`

Unsupported trailer of recovered data – only '`BC`' and '`CC`' values are supported [28, §8.1.2]

- `errLDS_AA_UnsupportedDigestAlgorithm`

Unsupported data hash algorithm [28, §7.3], [29]

- `errLDS_RI_SectorKey_CantFind`

It is impossible to find the sector public key data for RI (see section 5.8.17)

- `errLDS_RI_SectorKey_IncorrectData`

Incorrect sector public key data (see section 5.8.17)

- `errLDS_RI_SectorKey_IncompleteData`

All RI sector public key key parameters must be defined explicitly [24, part 3, §B.4.1]

- `errLDS_CV_Certificate_MissingMandatoryData_PK`

Absence of mandatory data fields in the structure of CV-certificate public key object [24, part 3, §C.1]

- `errLDS_CV_Certificate_PublicKey_Unsupported`

Unsupported public key format in the CV-certificate (for TA) (See also description of `errLDS_TA_PublicKey_UnsupportedAlgorithm`) [24, part 3, §A.6]

- `errLDS_CV_Certificate_CHAT_UnsupportedTerminalType`

Unsupported type of terminal in the CV-certificate [24, part 3, §C.4]

- `errLDS_CV_Certificate_PrivateKey_Unsupported`

Unsupported key algorithm in the terminal private key data (see section 4.9.3)

- `errLDS_CV_Certificate_PrivateKey_InvalidParams`

Incorrect terminal private key parameters data (see section 4.9.3)

- `errLDS_CV_Certificate_IncorrectData`

Incorrect format of CV-certificate TLV-data [24, part 3, §C.1].

- `errLDS_CV_Certificate_CPI_IncorrectData`

Incorrect format of the CV-certificate "Certificate Profile Identifier" field [24, part 3, §C.1].

- `errLDS_CV_Certificate_CAR_IncorrectData`

Incorrect format of the CV-certificate "Certificate Authority Reference" field [24, part 3, §C.1].

- `errLDS_CV_Certificate_PublicKey_IncorrectData`

Incorrect format of the CV-certificate public key data [24, part 3, §C.1].

- `errLDS_CV_Certificate_CHR_IncorrectData`

Incorrect format of the CV-certificate "Certificate Holder Reference" field [24, part 3, §C.1].

- `errLDS_CV_Certificate_CHAT_IncorrectData`

Incorrect format of the CV-certificate "Certificate Holder Authorization Template" field [24, part 3, §C.1].

- `errLDS_CV_Certificate_ValidFrom_IncorrectData`

Incorrect format of the CV-certificate "Certificate Effective Date" field [24, part 3, §C.1].

- `errLDS_CV_Certificate_ValidTo_IncorrectData`

Incorrect format of the CV-certificate "Certificate Expiration Date" field [24, part 3, §C.1].

- `errLDS_CV_Certificate_Extensions_IncorrectData`

Incorrect format of the CV-certificate extensions[24, part 3, §C.1].

- `errLDS_CV_Certificate_PrivateKey_IncorrectData`

Incorrect format of the private key data [10].

- `errLDS_CV_Certificate_PrivateKey_Missing`

Corresponding private key is not found for the terminal certificate (see sections 4.9.3, 5.5.3).

6.4.43. eLDS_ParsingNotificationCodes

eLDS_ParsingErrorCodes enumeration contains a set of codes of non-critical remarks detected during analysis of data structure used during SDK work (see section 5.2).

```
enum eLDS_ParsingNotificationCodes
{
   ntfLDS_ASN_Certificate_IncorrectVersion                   = 0x90000001,
   ntfLDS_ASN_Certificate_NonMatchingSignatureAlgorithm      = 0x90000002,
   ntfLDS_ASN_Certificate_IncorrectTimeCoding                = 0x90000003,
   ntfLDS_ASN_Certificate_IncorrectUseOfGeneralizedTime      = 0x90000004,
   ntfLDS_ASN_Certificate_EmptyIssuer                        = 0x90000005,
   ntfLDS_ASN_Certificate_EmptySubject                       = 0x90000006,
   ntfLDS_ASN_Certificate_UnsupportedCriticalExtension       = 0x90000008,
   ntfLDS_ASN_Certificate_ForcedDefaultCSCARole              = 0x9000000E,
   ntfLDS_ASN_Certificate_ForcedDefaultDSRole                = 0x9000000F,
   ntfLDS_ASN_Certificate_IncorrectIssuerSubjectDS           = 0x90000010,
   ntfLDS_ASN_Certificate_DuplicatingExtensions              = 0x90000017,

   ntfLDS_ICAO_Certificate_Version_Missed                    = 0x90000200,
   ntfLDS_ICAO_Certificate_Version_Incorrect                 = 0x90000201,
   ntfLDS_ICAO_Certificate_SN_NonCompliant                   = 0x90000241,
   ntfLDS_ICAO_Certificate_Issuer_Country_Missed             = 0x90000202,
   ntfLDS_ICAO_Certificate_Issuer_CommonName_Missed          = 0x90000203,
   ntfLDS_ICAO_Certificate_Issuer_CountryNonCompliant        = 0x90000204,
   ntfLDS_ICAO_Certificate_Issuer_SN_NonCompliant            = 0x90000242,
   ntfLDS_ICAO_Certificate_Issuer_AttributeNonCompliant      = 0x90000244,
   ntfLDS_ICAO_Certificate_Subject_Country_Missed            = 0x90000205,
   ntfLDS_ICAO_Certificate_Subject_CommonName_Missed         = 0x90000206,
   ntfLDS_ICAO_Certificate_Subject_CountryNonCompliant       = 0x90000207,
   ntfLDS_ICAO_Certificate_Subject_SN_NonCompliant           = 0x90000243,
   ntfLDS_ICAO_Certificate_Subject_AttributeNonCompliant     = 0x90000245,
   ntfLDS_ICAO_Certificate_IssuerSubject_Country_NonMatching = 0x90000246,

   ntfLDS_ICAO_Certificate_UsingNonCompliantData             = 0x90000208,
   ntfLDS_ICAO_Certificate_UnsupportedSignatureAlgorithm     = 0x90000209,
   ntfLDS_ICAO_Certificate_UnsupportedPublicKeyAlgorithm     = 0x9000020A,
   ntfLDS_ICAO_Certificate_MissedExtensions                  = 0x9000020B,
   ntfLDS_ICAO_Certificate_Validity                          = 0x9000020C,

   ntfLDS_ICAO_Certificate_Ext_UsingNonCompliantData         = 0x9000020D,
```

```
ntfLDS_ICAO_Certificate_Ext_KeyUsage_Missed             = 0x9000020E,
ntfLDS_ICAO_Certificate_Ext_KeyUsage_NotCritical        = 0x9000020F,
ntfLDS_ICAO_Certificate_Ext_KeyUsage_IncorrectData      = 0x90000210,

ntfLDS_ICAO_Certificate_Ext_BasicC_Missed               = 0x90000211,
ntfLDS_ICAO_Certificate_Ext_BasicC_IncorrectUsage1      = 0x90000212,
ntfLDS_ICAO_Certificate_Ext_BasicC_IncorrectUsage2      = 0x90000213,
ntfLDS_ICAO_Certificate_Ext_BasicC_NotCritical          = 0x90000214,
ntfLDS_ICAO_Certificate_Ext_BasicC_IncorrectData        = 0x90000215,
ntfLDS_ICAO_Certificate_Ext_BasicC_PathLenC_Missed      = 0x90000216,
ntfLDS_ICAO_Certificate_Ext_BasicC_PathLenC_Incorrect   = 0x90000217,

ntfLDS_ICAO_Certificate_Ext_ExtKeyUsage_NotCritical     = 0x90000218,
ntfLDS_ICAO_Certificate_Ext_ExtKeyUsage_IncorrectUsage  = 0x90000219,
ntfLDS_ICAO_Certificate_Ext_ExtKeyUsage_IncorrectData   = 0x9000021A,

ntfLDS_ICAO_Certificate_Ext_AuthKeyID_Missed            = 0x9000021B,
ntfLDS_ICAO_Certificate_Ext_AuthKeyID_IncorrectData     = 0x9000021C,
ntfLDS_ICAO_Certificate_Ext_AuthKeyID_KeyID_Missed      = 0x9000021D,

ntfLDS_ICAO_Certificate_Ext_SubjectKeyID_Missed         = 0x9000021E,
ntfLDS_ICAO_Certificate_Ext_SubjectKeyID_IncorrectData  = 0x9000021F,

ntfLDS_ICAO_Certificate_Ext_PrivateKeyUP_Missed         = 0x90000220,
ntfLDS_ICAO_Certificate_Ext_PrivateKeyUP_IncorrectData  = 0x90000221,
ntfLDS_ICAO_Certificate_Ext_PrivateKeyUP_Empty          = 0x90000222,

ntfLDS_ICAO_Certificate_Ext_SubjectAltName_Missed       = 0x90000223,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_IncorrectData= 0x90000224,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_Empty        = 0x90000225,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_NonCompliant = 0x90000226,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_Critical     = 0x90000228,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_DN_Empty     = 0x90000229,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_DN_Incorrect = 0x9000022A,
ntfLDS_ICAO_Certificate_Ext_SubjectAltName_DN_NonCompliant = 0x9000022B,

ntfLDS_ICAO_Certificate_Ext_IssuerAltName_Missed        = 0x9000022C,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_IncorrectData = 0x9000022D,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_Empty         = 0x9000022E,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_NonCompliant  = 0x9000022F,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_Critical      = 0x90000231,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_DN_Empty      = 0x90000232,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_DN_Incorrect  = 0x90000233,
ntfLDS_ICAO_Certificate_Ext_IssuerAltName_DN_NonCompliant= 0x90000234,


ntfLDS_ICAO_Certificate_Ext_CSCA_AltNames_NonMatching   = 0x90000247,

ntfLDS_ICAO_Certificate_Ext_NameChange_IncorrectData    = 0x90000248,
ntfLDS_ICAO_Certificate_Ext_NameChange_NonCompliant     = 0x90000249,
ntfLDS_ICAO_Certificate_Ext_NameChange_Critical         = 0x9000024A,

ntfLDS_ICAO_Certificate_Ext_DocTypeList_Missed          = 0x90000235,
ntfLDS_ICAO_Certificate_Ext_DocTypeList_IncorrectData   = 0x90000236,
ntfLDS_ICAO_Certificate_Ext_DocTypeList_Version         = 0x90000237,
ntfLDS_ICAO_Certificate_Ext_DocTypeList_DocTypes        = 0x90000238,
ntfLDS_ICAO_Certificate_Ext_DocTypeList_DocTypes_Empty  = 0x90000239,
```

```
ntfLDS_ICAO_Certificate_Ext_DocTypeList_NonCompliant     = 0x9000024B,
ntfLDS_ICAO_Certificate_Ext_DocTypeList_Critical         = 0x9000024C,

ntfLDS_ICAO_Certificate_Ext_CertPolicies_IncorrectData   = 0x9000023A,
ntfLDS_ICAO_Certificate_Ext_CertPolicies_Empty           = 0x9000023B,
ntfLDS_ICAO_Certificate_Ext_CertPolicies_PolicyID_Missed= 0x9000023C,

ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_Missed          = 0x9000023D,
ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_IncorrectData   = 0x9000023E,
ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_Empty           = 0x9000023F,
ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_PointMissed     = 0x90000240,

ntfLDS_ICAO_Certificate_Ext_Optional_Critical            = 0x9000024D,

ntfLDS_ICAO_COM_LDS_Version_Incorrect                    = 0x90000020,
ntfLDS_ICAO_COM_LDS_Version_Missing                      = 0x90000021,
ntfLDS_ICAO_COM_Unicode_Version_Incorrect                = 0x90000022,
ntfLDS_ICAO_COM_Unicode_Version_Missing                  = 0x90000023,
ntfLDS_ICAO_COM_DGPM_Incorrect                           = 0x90000024,
ntfLDS_ICAO_COM_DGPM_Missing                             = 0x90000025,
ntfLDS_ICAO_COM_DGPM_Unexpected                          = 0x90000026,

ntfLDS_ICAO_Application_LDSVersion_Unsupported           = 0x90000030,
ntfLDS_ICAO_Application_UnicodeVersion_Unsupported       = 0x90000031,
ntfLDS_ICAO_Application_LDSVersion_Inconsistent          = 0x90000032,
ntfLDS_ICAO_Application_UnicodeVersion_Inconsistent      = 0x90000033,

ntfLDS_ASN_SignedData_OID_Incorrect                      = 0x90000100,
ntfLDS_ASN_SignedData_Version_Incorrect                  = 0x900001A0,
ntfLDS_ASN_SignedData_ContentOID_Incorrect               = 0x900001A1,

ntfLDS_ICAO_SignedData_Version_Incorrect                 = 0x90000101,
ntfLDS_ICAO_SignedData_DigestAlgorithms_Empty            = 0x90000102,
ntfLDS_ICAO_SignedData_DigestAlgorithms_Unsupported      = 0x90000103,
ntfLDS_ICAO_SignedData_SignerInfos_MultipleEntries       = 0x90000109,

ntfLDS_ICAO_SignedData_Certificates_Missed               = 0x900001B0,
ntfLDS_ICAO_SignedData_Certificates_Empty                = 0x900001B1,
ntfLDS_ICAO_SignedData_CRLs_IncorrectUsage               = 0x900001B2,

ntfLDS_ICAO_LDSObject_IncorrectContentOID                = 0x90000104,
ntfLDS_ICAO_LDSObject_DGNumber_Incorrect                 = 0x90000105,
ntfLDS_ICAO_LDSObject_DGHash_Missing                     = 0x90000106,
ntfLDS_ICAO_LDSObject_DGHash_Extra                       = 0x90000107,
ntfLDS_ICAO_LDSObject_Version_Incorrect                  = 0x90000108,

ntfLDS_ICAO_MasterList_Version_Incorrect                 = 0x900001C0,
ntfLDS_ICAO_DeviationList_Version_Incorrect              = 0x900001C8,
ntfLDS_BSI_DefectList_Version_Incorrect                  = 0x900001D0,
ntfLDS_BSI_BlackList_Version_Incorrect                   = 0x900001D8,

ntfLDS_ASN_SignerInfo_Version_Incorrect                  = 0x9000010A,
ntfLDS_ASN_SignerInfo_SID_IncorrectChoice                = 0x9000010B,
ntfLDS_ASN_SignerInfo_SID_DigestAlgorithmNotListed       = 0x9000010C,
ntfLDS_ASN_SignerInfo_MessageDigestAttr_Missing          = 0x9000010D,
ntfLDS_ASN_SignerInfo_MessageDigestAttr_Data             = 0x9000010E,
ntfLDS_ASN_SignerInfo_MessageDigestAttr_Value            = 0x9000010F,
ntfLDS_ASN_SignerInfo_ContentTypeAttr_Missing            = 0x90000110,
```

```
ntfLDS_ASN_SignerInfo_ContentTypeAttr_Data                 = 0x90000111,
ntfLDS_ASN_SignerInfo_ContentTypeAttr_Value                = 0x90000112,
ntfLDS_ASN_SignerInfo_SigningTimeAttr_Missing              = 0x9000011B,
ntfLDS_ASN_SignerInfo_SigningTimeAttr_Data                 = 0x9000011C,
ntfLDS_ASN_SignerInfo_SigningTimeAttr_Value                = 0x9000011D,
ntfLDS_ASN_SignerInfo_ListContentDescriptionAttr_Missing   = 0x9000011E,
ntfLDS_ASN_SignerInfo_ListContentDescriptionAttr_Data      = 0x9000011F,

ntfLDS_Auth_SignerInfo_Certificate_Validity                = 0x90000115,
ntfLDS_Auth_SignerInfo_Certificate_RootIsNotTrusted        = 0x90000116,
ntfLDS_Auth_SignerInfo_Certificate_CantFindCSCA            = 0x90000117,
ntfLDS_Auth_SignerInfo_Certificate_Revoked                 = 0x90000118,
ntfLDS_Auth_SignerInfo_Certificate_SignatureInvalid        = 0x90000119,

ntfLDS_UnsupportedImageFormat                              = 0x9000011A,

ntfLDS_MRZ_DocumentType_Unknown                            = 0x00022008,
ntfLDS_MRZ_IssuingState_SyntaxError                        = 0x00022009,
ntfLDS_MRZ_Name_IsVoid                                     = 0x0002200A,
ntfLDS_MRZ_Number_IncorrectChecksum                        = 0x0002200D,
ntfLDS_MRZ_Nationality_SyntaxError                         = 0x0002200E,
ntfLDS_MRZ_DOB_SyntaxError                                 = 0x0002200F,
ntfLDS_MRZ_DOB_Error                                       = 0x00022010,
ntfLDS_MRZ_DOB_IncorrectChecksum                           = 0x00022011,
ntfLDS_MRZ_Sex_Incorrect                                   = 0x00022012,
ntfLDS_MRZ_DOE_SyntaxError                                 = 0x00022013,
ntfLDS_MRZ_DOE_Error                                       = 0x00022014,
ntfLDS_MRZ_DOE_IncorrectChecksum                           = 0x00022015,
ntfLDS_MRZ_OptionalData_IncorrectChecksum                  = 0x00022016,
ntfLDS_MRZ_IncorrectChecksum                               = 0x00022017,
ntfLDS_MRZ_Incorrect                                       = 0x00022018,

ntfLDS_Biometrics_FormatOwner_Missing                      = 0x90010000,
ntfLDS_Biometrics_FormatOwner_Incorrect                    = 0x90020000,
ntfLDS_Biometrics_FormatType_Missing                       = 0x90030000,
ntfLDS_Biometrics_FormatType_Incorrect                     = 0x90040000,
ntfLDS_Biometrics_Type_Incorrect                           = 0x90050000,
ntfLDS_Biometrics_SubType_Missing                          = 0x90060000,
ntfLDS_Biometrics_SubType_Incorrect                        = 0x90070000,
ntfLDS_Biometrics_BDB_Image_Missing                        = 0x90080000,
ntfLDS_Biometrics_BDB_FormatID_Incorrect                   = 0x90090000,
ntfLDS_Biometrics_BDB_Version_Incorrect                    = 0x900A0000,
ntfLDS_Biometrics_BDB_DataLength_Incorrect                 = 0x900B0000,

ntfLDS_Biometrics_BDB_Data_Gender                          = 0x90100000,
ntfLDS_Biometrics_BDB_Data_EyeColor                        = 0x90110000,
ntfLDS_Biometrics_BDB_Data_HairColor                       = 0x90120000,
ntfLDS_Biometrics_BDB_Data_PoseAngle_Yaw                   = 0x90130000,
ntfLDS_Biometrics_BDB_Data_PoseAngle_Pitch                 = 0x90140000,
ntfLDS_Biometrics_BDB_Data_PoseAngle_Roll                  = 0x90150000,
ntfLDS_Biometrics_BDB_Data_PoseAngleU_Yaw                  = 0x90160000,
ntfLDS_Biometrics_BDB_Data_PoseAngleU_Pitch                = 0x90170000,
ntfLDS_Biometrics_BDB_Data_PoseAngleU_Roll                 = 0x90180000,
ntfLDS_Biometrics_BDB_Data_FaceImageType                   = 0x90190000,
ntfLDS_Biometrics_BDB_Data_ImageDataType                   = 0x901A0000,
ntfLDS_SI_PACE_Info_UnsupportedStdParameters               = 0x91000000,
ntfLDS_SI_PACE_Info_DeprecatedVersion                      = 0x91000001,
```

```
   ntfLDS_SI_PACE_DomainParams_UsingStdRef                  = 0x91000002,
   ntfLDS_SI_PACE_DomainParams_UnsupportedAlgorithm         = 0x91000003,

   ntfLDS_SI_CA_Info_IncorrectVersion                       = 0x91000004,
   ntfLDS_SI_CA_PublicKey_UnsupportedAlgorithm              = 0x91000005,
   ntfLDS_SI_CA_DomainParams_UnsupportedAlgorithm           = 0x91000006,

   ntfLDS_SI_TA_Info_IncorrectVersion                       = 0x91000007,
   ntfLDS_SI_TA_Info_FileIDForVersion2                      = 0x91000008,

   ntfLDS_SI_eIDSecurity_UnsupportedDigestAlgorithm         = 0x91000009,

   ntfLDS_SI_RI_Info_IncorrectVersion                       = 0x9100000A,
   ntfLDS_SI_RI_DomainParams_UnsupportedAlgorithm           = 0x9100000B,

   ntfLDS_SI_AA_Info_IncorrectVersion                       = 0x9100000C,
   ntfLDS_SI_AA_Info_UnsupportedAlgorithm                   = 0x9100000D,
   ntfLDS_SI_AA_Info_InconsistantAlgorithmReference         = 0x9100000E,

   ntfLDS_SI_Storage_PACE_Info_NotAvailable                 = 0x91000100,
   ntfLDS_SI_Storage_PACE_Info_NoStdParameters              = 0x91000101,
   ntfLDS_SI_Storage_PACE_Info_NoMatchingDomainParams       = 0x91000102,

   ntfLDS_SI_Storage_CA_Info_NotAvailable                   = 0x91000103,
   ntfLDS_SI_Storage_CA_DomainParams_NoRequiredOption       = 0x91000104,
   ntfLDS_SI_Storage_CA_DomainParams_NotAvailable           = 0x91000105,
   ntfLDS_SI_Storage_CA_AnonymousInfos                      = 0x91000106,
   ntfLDS_SI_Storage_CA_Info_NoMatchingDomainParams         = 0x91000107,
   ntfLDS_SI_Storage_CA_Info_NoMatchingPublicKey            = 0x91000108,
   ntfLDS_SI_Storage_CA_IncorrectInfosQuantity              = 0x91000109,

   ntfLDS_SI_Storage_TA_Info_NotAvailable                   = 0x9100010A,

   ntfLDS_SI_Storage_CardInfoLocator_MultipleEntries        = 0x9100010B,
   ntfLDS_SI_Storage_eIDSecurityInfo_MultipleEntries        = 0x9100010C,
   ntfLDS_SI_Storage_PrivilegedTI_MultipleEntries           = 0x9100010D,
   ntfLDS_SI_Storage_PrivilegedTI_IncorrectUsage            = 0x9100010E,
   ntfLDS_SI_Storage_RI_DomainParams_MultipleEntries        = 0x9100010F,

   ntfLDS_SI_Storage_PACEInfos_NonConsistant                = 0x91000110,

   ntfLDS_CVCertificate_Profile_IncorrectVersion            = 0x91000201,
   ntfLDS_CVCertificate_Validity                            = 0x91000202,
   ntfLDS_CVCertificate_NonCVCADomainParameters             = 0x91000203,

   ntfLDS_CV_Certificate_PrivateKey_IncorrectVersion        = 0x91000204,

   ntfLDS_TA_PACEStaticBindingUsed                          = 0x91000300,

   ntfLDS_Auth_MLSignerInfo_Certificate_Validity            = 0x92000115,
   ntfLDS_Auth_MLSignerInfo_Certificate_RootIsNotTrusted    = 0x92000116,
   ntfLDS_Auth_MLSignerInfo_Certificate_CantFindCSCA        = 0x92000117,
   ntfLDS_Auth_MLSignerInfo_Certificate_Revoked             = 0x92000118,
   ntfLDS_Auth_MLSignerInfo_Certificate_SignatureInvalid    = 0x92000119,
};
```

Constants describe appearance of the following situations:

- ntfLDS_ASN_Certificate_IncorrectVersion

Incorrect version of the certificate – the value of version field of TBSCertificate ASN.1-object does not match the object contents [6, §4.1.2.1].

- ntfLDS_ASN_Certificate_NonMatchingSignatureAlgorithm

The contents of signature field of TBSCertificate ASN.1-object does not match the contents of signatureAlgorithm field of Certificate object [6, §4.1.2.3].

- ntfLDS_ASN_Certificate_IncorrectTimeCoding

Incorrect format of the contents of validity field of TBSCertificate ASN.1-object [6, §4.1.2.5]. Expected date format for UTCTime – YYMMDDHHMMSSZ, for Generalized-Time – YYYYMMDDHHMMSSZ.

- ntfLDS_ASN_Certificate_IncorrectUseOfGeneralizedTime

Incorrect format of the contents of validity field of TBSCertificate ASN.1-object [6, §4.1.2.5]. Date for the year <2050 should be coded using UTCTime [2, section IV, §A1.2].

- ntfLDS_ASN_Certificate_EmptyIssuer

The data of issuer field of TBSCertificate ASN.1-object must include at least one RelativeDistinguishedName element [6, §4.1.2.4].

- ntfLDS_ASN_Certificate_EmptySubject

The data of subject field of TBSCertificate ASN.1-object must include at least one attribute –RelativeDistinguishedName element [6, §4.1.2.6].

- ntfLDS_ASN_Certificate_UnsupportedCriticalExtension

Certificate critical extensions (extensions contents of TBSCertificate ASN.1-object) contain unsupported extensions [6, §4.2], [2, section IV, §A1.2].

- ntfLDS_ASN_Certificate_ForcedDefaultCSCARole

Role of self-signed certificate is not designated in the mandatory extensions as CSCA (keyUsage must contain keyCertSign, BasicConstraints – cA=true) [6, §4.2].

- ntfLDS_ASN_Certificate_ForcedDefaultDSRole

Role of non self-signed certificate is not designated in the mandatory extensions as DS (keyUsage must contain digitalSignature) [6, §4.2].

- ntfLDS_ASN_Certificate_IncorrectIssuerSubjectDS

Role of self-signed certificate is designated in the mandatory extensions incorrectly (keyUsage contains digitalSignature) [6, §4.2].

- ntfLDS_ASN_Certificate_DuplicatingExtensions

Found multiple copies of the same certificate extension [6, §4.2].

- ntfLDS_ICAO_Certificate_Version_Missed

Missing mandatory version field of TBSCertificate ASN.1-object [6, §4.1], [2, section IV, §A1.1].

- ntfLDS_ICAO_Certificate_Version_Incorrect

Incorrect value of version field of TBSCertificate ASN.1-object – should contain the number 2, indicating the version 2 certificate data [6, §4.1], [2, section IV, §A1.1].

- ntfLDS_ICAO_Certificate_SN_NonCompliant

Contents of serialNumber field of TBSCertificate ASN.1-object must be a positive integer of length no more than 20 octets and represented in the smallest number of octets [6, §4.1.2.2], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Issuer_Country_Missed

Missing mandatory Country attribute in issuer field of TBSCertificate ASN.1-object [6, §4.1.2.4], [2, section IV, §A1.5].

- ntfLDS_ICAO_Certificate_Issuer_CommonName_Missed

Missing mandatory CommonName attribute in issuer field of TBSCertificate ASN.1-object [6, §4.1.2.4], [2, section IV, §A1.5].

- ntfLDS_ICAO_Certificate_Issuer_CountryNonCompliant

The country code in CountryName attribute in issuer field of TBSCertificate ASN.1-object should be ecoded in two uppercase alphabetic ASCII-characters [6, §4.1.2.4, §4.1.2.6], [2, section IV, §A1.5], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Issuer_SN_NonCompliant

SerialNumber attribute in issuer field of TBSCertificate ASN.1-object must be PrintableString [6, §4.1.2.4], [2, section IV, §A1.5], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Issuer_AttributeNonCompliant

Attributes in issuer field of TBSCertificate ASN.1-object must be PrintableString or UTF8String [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Subject_Country_Missed

Missing mandatory Country attribute in subject field of TBSCertificate ASN.1-object [6, §4.1.2.4], [2, section IV, §A1.5].

- ntfLDS_ICAO_Certificate_Subject_CommonName_Missed

Missing mandatory CommonName attribute in subject field of TBSCertificate ASN.1-object [6, §4.1.2.4], [2, section IV, §A1.5].

- ntfLDS_ICAO_Certificate_Subject_CountryNonCompliant

The country code in CountryName attribute in subject field of TBSCertificate ASN.1-object should be ecoded in two uppercase alphabetic ASCII-characters [6, §4.1.2.4, §4.1.2.6], [2, section IV, §A1.5], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Subject_SN_NonCompliant

SerialNumber attribute in subject field of TBSCertificate ASN.1-object must be PrintableString [6, §4.1.2.6], [2, section IV, §A1.5], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Subject_AttributeNonCompliant

Attributes in issuer field of TBSCertificate ASN.1-object must be Printable-String or UTF8String [31, §3.2.1].

- ntfLDS_ICAO_Certificate_IssuerSubject_Country_NonMatching

CountryName attributes in issuer and subject fields of TBSCertificate ASN.1-object must be identical [31, §3.2.1].

- ntfLDS_ICAO_Certificate_UsingNonCompliantData

TBSCertificate ASN.1-object contains unauthorized by ICAO elements (issuerUniqueID or subjectUniqueID) [2, section IV, § A1.1].

- ntfLDS_ICAO_Certificate_UnsupportedSignatureAlgorithm

signatureAlgorithm field of Certificate ASN.1-object contains an identifier (OID) of the unsupported digital signature algorithm [6, §4.1], [5, §2.2].

- ntfLDS_ICAO_Certificate_UnsupportedPublicKeyAlgorithm

subjectPublicKeyInfo field of TBSCertificate ASN.1-object contains an identifier (OID) of the unsupported public key algorithm [6, §4.1], [5, §2.3].

- ntfLDS_ICAO_Certificate_MissedExtensions

Certificate critical extensions (extensions contents of TBSCertificate ASN.1-object) do not contain all required elements [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Validity

The certificate has expired, or its action has not yet begun.

- ntfLDS_ICAO_Certificate_Ext_UsingNonCompliantData

The list of extensions of TBSCertificate ASN.1-object contains elements that are not allowed to use [6, §4.1], [2, section IV, §A1. 2], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_KeyUsage_Missed

Missing a mandatory KeyUsage certificate extension [6, §4.2.1.3], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_KeyUsage_NotCritical

Mandatory KeyUsage certificate extension is not marked as critical [6, §4.2.1.3], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_KeyUsage_IncorrectData

Incorrect ASN.1-data of KeyUsage certificate extension [6, §4.2.1.3], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_BasicC_Missed

Missing a mandatory BasicConstraints certificate extension [6, §4.2.1.9], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_BasicC_IncorrectUsage1

The erroneous use of BasicConstraints certificate extension – use for DS-certificate [6, §4.2.1.9], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_BasicC_IncorrectUsage2

The erroneous use of BasicConstraints certificate extension – not set a mandatory cA feature in combination with keyCertSign flag in KeyUsage extension [6, §4.2.1.9], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_BasicC_NotCritical

Mandatory BasicConstraints certificate extension is not marked as critical [6, §4.2.1.9], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_BasicC_IncorrectData

Incorrect ASN.1-data of BasicConstraints certificate extension [6, §4.2.1.9].

- ntfLDS_ICAO_Certificate_Ext_BasicC_PathLenC_Missed

Missing a mandatory pathLenConstraint field of BasicConstraints certificate extension [6, §4.2.1.9], [2, section IV, §A1.2].

- ntfLDS_ICAO_Certificate_Ext_BasicC_PathLenC_Incorrect

Incorrect pathLenConstraint value of BasicConstraints certificate extension – should contain 0 [6, §4.2.1.9], [35, R3-p1_v2_sIV_0038].

- ntfLDS_ICAO_Certificate_Ext_ExtKeyUsage_NotCritical

ExtKeyUsage certificate extension is present, but not marked as critical [6, §4.2.1.12], [31, §3.2.1], [34, §3.2].

- ntfLDS_ICAO_Certificate_Ext_ExtKeyUsage_IncorrectUsage

The erroneous use of ExtKeyUsage certificate extension – use for CSCA- or DS-certificate [6, §4.2.1.12], [31, §3.2.1], [34, §3.2].

- ntfLDS_ICAO_Certificate_Ext_ExtKeyUsage_IncorrectData

Incorrect ASN.1-data of ExtKeyUsage certificate extension [6, §4.2.1.12].

- ntfLDS_ICAO_Certificate_Ext_AuthKeyID_Missed

Missing a mandatory AuthorityKeyIdentifier certificate extension [2, section IV, §A1.2], [6, §4.2.1.1], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_AuthKeyID_IncorrectData

Incorrect ASN.1-data of AuthorityKeyIdentifier certificate extension [6, §4.2.1.1].

- ntfLDS_ICAO_Certificate_Ext_AuthKeyID_KeyID_Missed

Missing a mandatory keyIdentifier field of AuthorityKeyIdentifier certificate extension [6, §4.2.1.1], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectKeyID_Missed

Missing a mandatory SubjectKeyIdentifier certificate extension [6, §4.2.1.2], [2, section IV, §A1.2], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectKeyID_IncorrectData

Incorrect ASN.1-data of SubjectKeyIdentifier certificate extension [6, §4.2.1.2].

- ntfLDS_ICAO_Certificate_Ext_PrivateKeyUP_Missed

Missing a mandatory PrivateKeyUsagePeriod certificate extension [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_PrivateKeyUP_IncorrectData

Incorrect ASN.1-data of PrivateKeyUsagePeriod certificate extension [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_PrivateKeyUP_Empty

Empty PrivateKeyUsagePeriod certificate extension [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_Missed

Missing a mandatory SubjectAltName certificate extension [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_IncorrectData

Incorrect ASN.1-data of SubjectAltName certificate extension [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_Empty

Empty SubjectAltName certificate extension [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_NonCompliant

A set of used fields in SubjectAltName certificate extension not correspond to the requirements of ICAO [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_Critical

SubjectAltName certificate extension must not be marked as critical [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_DN_Empty

Empty directoryName field in SubjectAltName certificate extension [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_DN_Incorrect

Missing a mandatory localityName attribute in directoryName field of SubjectAltName certificate extension [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_SubjectAltName_DN_NonCompliant

The use of unauthorized attributes (apart from localityName and stateOrProvinceName) in directoryName field of SubjectAltName certificate extension [6, §4.2.1.6], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_Missed

Missing a mandatory IssuerAltName certificate extension [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_IncorrectData

Incorrect ASN.1-data of IssuerAltName certificate extension [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_Empty

Empty IssuerAltName certificate extension [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_NonCompliant

A set of used fields in IssuerAltName certificate extension not correspond to the requirements of ICAO [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_Critical

IssuerAltName certificate extension must not be marked as critical [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_DN_Empty

Empty directoryName field in IssuerAltName certificate extension [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_DN_Incorrect

Missing a mandatory localityName attribute in directoryName field of IssuerAltName certificate extension [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_IssuerAltName_DN_NonCompliant

The use of unauthorized attributes (apart from localityName and stateOrProvinceName) in directoryName field of IssuerAltName certificate extension [6, §4.2.1.7], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CSCA_AltNames_NonMatching

Contents of SubjectAltName and IssuerAltName fields of CSCA-certificate must be identical [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_NameChange_IncorrectData

Incorrect ASN.1-data of nameChange certificate extension [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_NameChange_NonCompliant

nameChange certificate extension allowed for CSCA-link-certificates only [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_NameChange_Critical

nameChange certificate extension must not be marked as critical [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_DocTypeList_Missed

Missing a mandatory documentTypeList certificate extension [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_DocTypeList_IncorrectData

Incorrect ASN.1-data of documentTypeList certificate extension [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_DocTypeList_Version

Incorrect version value of documentTypeList certificate extension – should contain 0 [31, §3.2.2].

- ntfLDS_ASN_Certificate_IncorrectIssuerSubjectDS

Role of self-signed certificate is designated in the mandatory extensions incorrectly (keyUsage contains digitalSignature) [6, §4.2].

- ntfLDS_ICAO_Certificate_Ext_DocTypeList_DocTypes

Incorrect content of docTypeList list items of documentTypeList certificate extension – should contain one- or two-letter codes for document types [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_DocTypeList_DocTypes_Empty

Empty docTypeList list of documentTypeList certificate extension [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_DocTypeList_NonCompliant

docTypeList certificate extension allowed for DS-certificates only [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_NameChange_Critical

docTypeList certificate extension must not be marked as critical [31, §3.2.2].

- ntfLDS_ICAO_Certificate_Ext_CertPolicies_IncorrectData

Incorrect ASN.1-data of certificatePolicies certificate extension [6, §4.2.1.4], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CertPolicies_Empty

Empty certificatePolicies certificate extension [6, §4.2.1.4], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CertPolicies_PolicyID_Missed

Missing a mandatory policyIdentifier field of certificatePolicies certificate extension [6, §4.2.1.4], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_Missed

Missing a mandatory CRLDistributionPoints certificate extension [6, §4.2.1.13], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_IncorrectData

Incorrect ASN.1-data of CRLDistributionPoints certificate extension [6, §4.2.1.13], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_Empty

Empty CRLDistributionPoints certificate extension [6, §4.2.1.13], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_CRLDistPoint_PointMissed

Empty element in CRLDistributionPoints certificate extension [6, §4.2.1.13], [31, §3.2.1].

- ntfLDS_ICAO_Certificate_Ext_Optional_Critical

Unsupported critical certificate extension usage [31, §3.2.1].

- ntfLDS_ICAO_COM_LDS_Version_Incorrect

Incorrect format or contents of the *LDS version* in EF.COM [2, section III, §14, §A.13.1]. Version 1.7 is expected (value '0170').

- ntfLDS_ICAO_COM_LDS_Version_Missing

Missing *LDS version* field in EF.COM [2, section III, §14, §A.13.1].

- ntfLDS_ICAO_COM_Unicode_Version_Incorrect

Incorrect format or contents of the *Unicode version* in EF.COM [2, section III, § 14, §A.13.1].

- ntfLDS_ICAO_COM_Unicode_Version_Missing

Missing *Unicode version* field in EF.COM [2, section III, § 14, §A.13.1].

- ntfLDS_ICAO_COM_DGPM_Incorrect

Incorrect format or contents of the *Data Group Presence Map* (*DGPM*) in EF.COM [2, section III, §14, §A.13.1].

- ntfLDS_ICAO_COM_DGPM_Missing

Missing *DGPM* field in EF.COM [2, section III, § 14, §A.13.1].

- ntfLDS_ICAO_COM_DGPM_Unexpected

Incorrect *DGPM* contents in EF.COM [2, section III, § 14, §A.13.1].

- ntfLDS_ICAO_Application_LDSVersion_Unsupported

Unsupported LDS version, registered in EF.COM or EF.SOD [2, section III, §14, §A.13.1], [31, §2.2].

- ntfLDS_ICAO_Application_UnicodeVersion_Unsupported

Unsupported Unicode version, registered in EF.COM or EF.SOD [2, section III, §14, §A.13.1], [31, §2.2].

- ntfLDS_ICAO_Application_LDSVersion_Inconsistent

LDS version mismatch, registered in EF.COM and EF.SOD [2, section III, §14, §A.13.1], [31, §2.2].

- ntfLDS_ICAO_Application_UnicodeVersion_Inconsistent

Unicode version mismatch, registered in EF.COM and EF.SOD [2, section III, §14, §A.13.1], [31, §2.2].

- ntfLDS_ASN_SignedData_OID_Incorrect

Incorrect identifier in contentType field of ContentInfo ASN.1-object, containing data of document security object. szOID_RSA_signedData identifier is expected, which defines the contents as SignedData [7, §3, §5.1].

- ntfLDS_ASN_SignedData_Version_Incorrect

Incorrect version value of SignedData ASN.1-object, containing data of document security object. Valid values are 1, 3, 4, 5 [7, §3, §5.1].

- ntfLDS_ASN_SignedData_ContentOID_Incorrect

Incorrect identifier in encapContentInfo.eContentType field of SignedData ASN.1-object, containing data of document security object. szOID_ICAO_MRTD_Security_LDSSecurityObject or szOID_LDSSecurityObject identifiers are expected, which defines the contents as LDSSecurityObject [2, section IV, §A3.2].

- `ntfLDS_ICAO_SignedData_Version_Incorrect`

Incorrect value in `version` field of `SignedData` ASN.1-object, containing data of the document security object [7, §5.1]. Value `3` is expected [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- `ntfLDS_ICAO_SignedData_DigestAlgorithms_Empty`

Empty list of hash algorithms in `digestAlgorithms` field of `SignedData` ASN.1-object, containing data of the document security object [7, §5.1], [2, section IV, §A3.1], [24, part 3, §A.1.2.5].

- `ntfLDS_ICAO_SignedData_DigestAlgorithms_Unsupported`

`digestAlgorithms` list of `SignedData` ASN.1-object contains identifiers of unsupported hash algorithms [7, §5.1], [2, section IV, § A3.1], [24, part 3, §A.1.2.5].

- `ntfLDS_ICAO_SignedData_SignerInfos_MultipleEntries`

Multiple digital signature data objects found in the document security object (in `signerInfos` list of `SignedData` ASN.1-object [7, §5.1]). It is expected the presence of a single element [2, section IV, §A3.1].

- `ntfLDS_ICAO_SignedData_Certificates_Missed`

Missing `certificates` list in `SignedData` ASN.1-object [7, §5.1], [2, section IV, §A3.1].

- `ntfLDS_ICAO_SignedData_Certificates_Empty`

Empty `certificates` list in `SignedData` ASN.1-object [7, §5.1], [2, section IV, §A3.1].

- `ntfLDS_ICAO_SignedData_CRLs_IncorrectUsage`

The presence of unauthorized `crls` list in `SignedData` ASN.1-object [7, §5.1], [2, section IV, §A3.1].

- `ntfLDS_ICAO_LDSObject_IncorrectContentOID`

Incorrect format of the contents of `encapContentInfo` field of `SignedData` ASN.1-object [7, §5.1] for the document security object from `EF.SOD` (see section 4.3.1). `LDSSecurityObject` object should be used to indicate `encapContentInfo` contents [2, section IV, §A3.2].

- `ntfLDS_ICAO_LDSObject_DGNumber_Incorrect`

`dataGroupHashValues` list of `LDSSecurityObject` ASN.1-object contains incorrect data group identifiers [2, section IV, §A3.2]. Values in the range 1-16 are allowed.

- `ntfLDS_ICAO_LDSObject_DGHash_Missing`

`dataGroupHashValues` list of `LDSSecurityObject` ASN.1-object contains incomplete set of data group identifiers, presence of which is defined by the contents of `EF.COM` [2, section IV, §A3.2].

- `ntfLDS_ICAO_LDSObject_DGHash_Extra`

`dataGroupHashValues` list of `LDSSecurityObject` ASN.1-object contains data group identifier not present in `EF.COM` [2, section IV, §A3.2].

- ntfLDS_ICAO_LDSObject_Version_Incorrect

Incorrect value in version field of LDSSecurityObject ASN.1-object. The 0 value is expected [2, section IV, §A3.2].

- ntfLDS_ICAO_MasterList_Version_Incorrect

Incorrect value in version field of CscaMasterList ASN.1-object. The 0 value is expected [34, §3.1.2].

- nfLDS_ICAO_DeviationList_Version_Incorrect

Incorrect value in version field of DeviationList ASN.1-object. The 0 value is expected [34, §3.1.2].

- ntfLDS_BSI_DefectList_Version_Incorrect

Incorrect value in version field of DefectList ASN.1-object. The 0 value is expected [36, A.1].

- ntfLDS_BSI_BlackList_Version_Incorrect

Incorrect value in version field of BlackList ASN.1-object. The 0 value is expected [36, B.1].

- ntfLDS_ASN_SignerInfo_Version_Incorrect

version field of SignerInfo ASN.1-object with the data of document security object digital signature [7, § 5.3] contains incorrect value. 1 and 3 values are allowed.

- ntfLDS_ASN_SignerInfo_SID_IncorrectChoice

Incompliance between version value and the choice of representation of the contents of sid field of SignerInfo ASN.1-object with data of document security objectdigital signature [7, §5.3].

- ntfLDS_ASN_SignerInfo_SID_DigestAlgorithmNotListed

digestAlgorithm from SignerInfo is not included in digestAlgorithms list of SignedData object [7, §5.3].

- ntfLDS_ASN_SignerInfo_MessageDigestAttr_Missing

signedAttrs list of attributes of SignerInfo ASN.1-object contains no MessageDigest element [7, §5.3, §11.2].

- ntfLDS_ASN_SignerInfo_MessageDigestAttr_Data

Incorrect data format of MessageDigest element in signedAttrs list of attributes of SignerInfo ASN.1-object [7, §5.3, §11.2].

- ntfLDS_ASN_SignerInfo_MessageDigestAttr_Value

Incorrect contents of MessageDigest element in signedAttrs list of attributes of SignerInfo ASN.1-object [7, §5.3, §11.2] (when compared with the actual hash value).

- ntfLDS_ASN_SignerInfo_ContentTypeAttr_Missing

signedAttrs list of attributes of SignerInfo ASN.1-object contains no ContentType element [7, §5.3, §11.1].

- ntfLDS_ASN_SignerInfo_ContentTypeAttr_Data

Incorrect data format of ContentType element in signedAttrs list of attributes of SignerInfo ASN.1-object [7, §5.3, §11.1].

- ntfLDS_ASN_SignerInfo_ContentTypeAttr_Value

Incorrect contents of ContentType element in signedAttrs list of attributes of SignerInfo ASN.1-object (must contain a value from SignedData.encapContentInfo.eContentType) [7, §5.3, §11.1].

- ntfLDS_ASN_SignerInfo_SigningTimeAttr_Missing

signedAttrs list of attributes of SignerInfo ASN.1-object contains no SigningTime element [7, §5.3, §11.3], [34, §3.1.1].

- ntfLDS_ASN_SignerInfo_SigningTimeAttr_Data

Incorrect data format of SigningTime element in signedAttrs list of attributes of SignerInfo ASN.1-object [7, §5.3, §11.3], [34, §3.1.1].

- ntfLDS_ASN_SignerInfo_SigningTimeAttr_Value

Incorrect contents of SigningTime element in signedAttrs list of attributes of SignerInfo ASN.1-object [7, §5.3, §11.3], [34, §3.1.1].

- ntfLDS_ASN_SignerInfo_ListContentDescriptionAttr_Missing

signedAttrs list of attributes of SignerInfo ASN.1-object for DeviationList/BlackList contains no ListContentDescription element [36, C.1].

- ntfLDS_ASN_SignerInfo_ListContentDescriptionAttr_Data

Incorrect data format of ListContentDescription element in signedAttrs list of attributes of SignerInfo ASN.1-object [36, C.1].

- ntfLDS_Auth_SignerInfo_Certificate_Validity

DS-certificate is expired (see section 5.5.2).

- ntfLDS_Auth_SignerInfo_Certificate_RootIsNotTrusted

No trust to the source of the DS-certificate (see section 5.5.2).

- ntfLDS_Auth_SignerInfo_Certificate_CantFindCSCA

CSCA-certificate not found (see section 5.5.2).

- ntfLDS_Auth_SignerInfo_Certificate_Revoked

Certificate revoked (see section 5.5.2).

- ntfLDS_Auth_SignerInfo_Certificate_SignatureInvalid

DS-certificate signature verification failed (see sections 4.8.1, 5.8.12).

- ntfLDS_UnsupportedImageFormat

Unrecognized graphic field image format.

- ntfLDS_MRZ_DocumentType_Unknown

EF.DG1 contents remark – unknown document type.

- ntfLDS_MRZ_IssuingState_SyntaxError

EF.DG1 contents remark – document issuing country field syntax error.

- ntfLDS_MRZ_Name_IsVoid

EF.DG1 contents remark – empty name field.

- ntfLDS_MRZ_Number_IncorrectChecksum

EF.DG1 contents remark – incorrect document number field check sum.

- ntfLDS_MRZ_Nationality_SyntaxError

EF.DG1 contents remark – nationality field syntax error.

- ntfLDS_MRZ_DOB_SyntaxError

EF.DG1 contents remark – date of birth field syntax error.

- ntfLDS_MRZ_DOB_Error

EF.DG1 contents remark – date of birth field error.

- ntfLDS_MRZ_DOB_IncorrectChecksum

EF.DG1 contents remark – incorrect date of birth field check sum.

- ntfLDS_MRZ_Sex_Incorrect

EF.DG1 contents remark – not allowed contents of the field “Sex”.

- ntfLDS_MRZ_DOE_SyntaxError

EF.DG1 contents remark – date of expiry syntax error.

- ntfLDS_MRZ_DOE_Error

EF.DG1 contents remark – date of expiry field error.

- ntfLDS_MRZ_DOE_IncorrectChecksum

EF.DG1 contents remark – incorrect date of expiry field check sum.

- ntfLDS_MRZ_OptionalData_IncorrectChecksum

EF.DG1 contents remark – incorrect optional data field check sum.

- ntfLDS_MRZ_IncorrectChecksum

EF.DG1 contents remark – incorrect overall MRZ check sum.

- ntfLDS_MRZ_Incorrect

Additional full check of MRZ contents failed (see section 5.8.6).

- `ntfLDS_Biometrics_FormatOwner_Missing`

Mandatory `FormatOwner` field absent in the BHT (see sections 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_FormatOwner_Incorrect`

Incorrect contents of `FormatOwner` field of the BHT (see sections 5.8.10, 6.3.22). `fown-ISO_IEC_JTC_1_SC_37` value is expected (see section 6.4.19). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_FormatType_Missing`

Mandatory field with the description of the record format is absent in the BHT data (see sections 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_FormatType_Incorrect`

Biometric record format given in the BHT does not correspond to the type of the biometric data group (see section 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_Type_Incorrect`

Biometric data type of the BHT does not correspond to the type of the biometric data group (see section 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_SubType_Missing` – mandatory field with the description

Of biometric data subtype is absent in the BHT data (see sections 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_SubType_Incorrect`

BHT contains incorrect value of biometric data subtype (see sections 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Image_Missing`

BHT or image objects not found in the data of the information group. Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_FormatID_Incorrect`

Text format identifier does not correspond to the type of biometric data group (see sections 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Version_Incorrect`

Unsupported biometric data format version (see sections 5.8.10, 6.3.22). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_DataLength_Incorrect`

Incorrect length of biometric data. Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Data_Gender`

Incorrect gender of the DO in the biometric data (see sections 5.8.10, 6.3.25). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Data_EyeColor`

Incorrect eye color of the DO in the biometric data (see sections 5.8.10, 6.3.25). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Data_HairColor`

Incorrect hair color of the DO in the biometric data (see sections 5.8.10, 6.3.25). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values)

- `ntfLDS_Biometrics_BDB_Data_PoseAngle_Yaw`,
  `ntfLDS_Biometrics_BDB_Data_PoseAngle_Pitch`,
  `ntfLDS_Biometrics_BDB_Data_PoseAngle_Roll`

Incorrect pose in the biometric data (see sections 5.8.10, 6.3.26). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Data_PoseAngleU_Yaw`,
  `ntfLDS_Biometrics_BDB_Data_PoseAngleU_Pitch`,
  `ntfLDS_Biometrics_BDB_Data_PoseAngleU_Roll`

Incorrect pose of uncertaincy in the biometric data (see sections 5.8.10, 6.3.26). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Data_FaceImageType`

Incorrect image type in the biometric data (see sections 5.8.10, 6.3.28). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- `ntfLDS_Biometrics_BDB_Data_ImageDataType`

Incorrect image data type in the biometric data (see sections 5.8.10, 6.3.28). Low order `WORD` of the notification code contains the identifier of the biometric data source file (one of **`eRFID_DataFile_Type`** values).

- ntfLDS_SI_PACE_Info_UnsupportedStdParameters

PACEInfo refers to the unsupported set of standard public key parameters [23, §3.1.1], [24, part 3, §A.1.1.1].

- ntfLDS_SI_PACE_Info_DeprecatedVersion

Deprecated version in PACEInfo (value 2 is expected) [23, §3.1.3], [24, part 3, §A.1.1.1].

- ntfLDS_SI_PACE_DomainParams_UsingStdRef

PACEDomainParameterInfo should not contain references to the standard public key parameters [23, §5.3.2, §5.7.1], [24, part 3, §A.2.1].

- ntfLDS_SI_PACE_DomainParams_UnsupportedAlgorithm

Unsupported public key algorithm in domainParameter field of PACEDomainParameterInfo object [23, §5.3.2], [24, part 3, §A.2.1].

- ntfLDS_SI_CA_Info_IncorrectVersion

Unsupported CA version in version field of ChipAuthenticationInfo object [24, part 3, §A.1.1.2]. Values 1 or 2 are allowed.

- ntfLDS_SI_CA_PublicKey_UnsupportedAlgorithm

Unsupported CA public key algorithm in publicKey field of ChipAuthenticationPublicKeyInfo object [24, part 3, §A.1.1.2, §A.4].

- ntfLDS_SI_CA_DomainParams_UnsupportedAlgorithm

Unsupported CA public key algorithm parameters in domainParameter field of ChipAuthenticationDomainParameterInfo object [24, part 3, §A.1.1.2, §A.4].

- ntfLDS_SI_TA_Info_IncorrectVersion

Unsupported TA version in version field of TerminalAuthenticationInfo object [24, part 3, §A.1.1.3]. Values 1 or 2 are allowed.

- ntfLDS_SI_TA_Info_FileIDForVersion2

Use of efCVCA field in TerminalAuthenticationInfo object for TA, version 2 [24, part 3, §A.1.1.3]. Presence of efCVCA is allowed only for the version 1.

- ntfLDS_SI_eIDSecurity_UnsupportedDigestAlgorithm

Unsupported hash algorithm in eIDSecurityInfo object [24, part 3, §A.1.1.6].

- ntfLDS_SI_RI_Info_IncorrectVersion

Unsupported RI version in version field of RestrictedIdentificationInfo object [24, part 3, §A.1.1.4]. Only the value 1 is allowed.

- ntfLDS_SI_RI_DomainParams_UnsupportedAlgorithm

Unsupported RI public key algorithm parameters in domainParameter field of RestrictedIdentificationDomainParameterInfo object [24, part 3, §A.1.1.4].

- ntfLDS_SI_AA_Info_IncorrectVersion

For LDS version 1.8 and higher. Unsupported AA version in `version` field of `ActiveAuthenticationInfo` object [31, §5.2.2]. Value `1` is allowed.

- `ntfLDS_SI_AA_Info_UnsupportedAlgorithm`

For LDS version 1.8 and higher. Unsupported digital signature algorithm in `signatureAlgorithm` field of `ActiveAuthenticationInfo` object [31, §5.2.2].

- `ntfLDS_SI_AA_Info_InconsistantAlgorithmReference`

For LDS version 1.8 and higher. Inconsistance between AA digital signature algorithm from `signatureAlgorithm` field of `ActiveAuthenticationInfo` object [31, §5.2.2] and AA public key algorithm [2, §A4].

- `ntfLDS_SI_Storage_PACE_Info_NotAvailable`

No `PACEInfo` was found in `SecurityInfos` data [23, §5.3], [24, part 3, §A.1.1.1].

- `ntfLDS_SI_Storage_PACE_Info_NoStdParameters`

No `PACEInfo` referring to the standard set of public key parameters was found in `SecurityInfos` data [23, §5.3], [24, part 3, §A.1.1.1].

- `ntfLDS_SI_Storage_PACE_Info_NoMatchingDomainParams`

No `PACEDomainParameterInfo` corresponding to `PACEInfo`, referring to the non-standard set of public key parameters, was found in `SecurityInfos` data [23, §3.1.1], [24, part 3, §A.1.1.1].

- `ntfLDS_SI_Storage_CA_Info_NotAvailable`

No `ChipAuthenticationInfo` was found (for version 2) [24, part 3, §A.1.1.2, §A.1.2.1].

- `ntfLDS_SI_Storage_CA_DomainParams_NoRequiredOption`

No `ChipAuthenticationDomainParameterInfo`, referring to the standard set of public key parameters – for version 2, or to the explicitly defined public key parameters – for version 1, was found in `SecurityInfos` data [24, part 3, §A.2.1.3].

- `ntfLDS_SI_Storage_CA_DomainParams_NotAvailable`

No `ChipAuthenticationDomainParameterInfo` (for version 2) was found in `SecurityInfos` data [24, part 3, §A.1.1.2, §A.1.2.1].

- `ntfLDS_SI_Storage_CA_AnonymousInfos`

If several CA keys supported `keyId` field must be used in all `SecurityInfos` [24, part 3, §A.1.1.2].

- `ntfLDS_SI_Storage_CA_Info_NoMatchingDomainParams`

No `ChipAuthenticationDomainParameterInfo`, corresponding to `ChipAuthenticationInfo` object by `keyId` identifier, was found in `SecurityInfos` data [24, part 3, §A.1.2.1].

- `ntfLDS_SI_Storage_CA_Info_NoMatchingPublicKey`

No `ChipAuthenticationPublicKeyInfo`, corresponding to `ChipAuthentication-`

Info by keyId identifier, object was found in SecurityInfos data [24, part 3, §A.1.2.2, §A.1.2.3].

- ntfLDS_SI_Storage_CA_IncorrectInfosQuantity

Different number of ChipAuthenticationPublicKeyInfo and ChipAuthenticationInfo objects [24, part 3, §A.1.2.2, §A.1.2.3]. There must be a strict correspondence.

- ntfLDS_SI_Storage_TA_Info_NotAvailable

No TerminalAuthenticationInfo was found (for version 2) in SecurityInfos data [24, part 3, §A.1.1.3, §A.1.2.1].

- ntfLDS_SI_Storage_CardInfoLocator_MultipleEntries

Multiple CardInfoLocator objects were found in SecurityInfos data [24, part 3, §A.1.1.5]. Only one is allowed.

- ntfLDS_SI_Storage_eIDSecurityInfo_MultipleEntries

Multiple eIDSecurityInfo objects [24, part 3, §A.1.1.6] were found in SecurityInfos data. Only one is allowed.

- ntfLDS_SI_Storage_PrivilegedTI_MultipleEntries

Multiple PrivilegedTerminalInfo objects [24, part 3, § A.1.1.7] were found in SecurityInfos data. Only one is allowed.

- ntfLDS_SI_Storage_PrivilegedTI_IncorrectUsage

PrivilegedTerminalInfo objects usage in EF.CardSecurity is not allowed [24, part 3, §A.1.2.2].

- ntfLDS_SI_Storage_RI_DomainParams_MultipleEntries

Multiple RestrictedIdentificationDomainParameterInfo objects [24, part 3, §A.1.1.4] were found in SecurityInfos data. Only one is allowed.

- ntfLDS_SI_Storage_PACEInfos_NonConsistant

Non-consistant EF.CardAcess and DG14 [23, §5.4].

- ntfLDS_CVCertificate_Profile_IncorrectVersion

Unsupported version of CV-certificate profile [24, part 3, §C.1]. Only the value 0 is allowed.

- ntfLDS_CVCertificate_Validity

CV-certificate is expired.

- ntfLDS_CVCertificate_NonCVCADomainParameters

DV-certificates and terminal certificates must not contain explicitly defined EC public key parameters [24, part 3, §D.3.3].

- `ntfLDS_CV_Certificate_PrivateKey_IncorrectVersion`

Incorrect version of the private key data for the terminal authentication procedure [10] (see section 5.5.3).

- `ntfLDS_TA_PACEStaticBindingUsed`

Terminal authentication procedure was carried out using an algorithm with a static binding with PACE [24, part 1, §3.5] (see section 5.8.15).

- `ntfLDS_Auth_MLSignerInfo_Certificate_Validity`

MLS-certificate is expired (see section 5.5.2). Marks all certificates contained in the checked master list.

- `ntfLDS_Auth_MLSignerInfo_Certificate_RootIsNotTrusted`

No trust to the source of the MLS-certificate (see section 5.5.2). Marks all certificates contained in the checked master list.

- `ntfLDS_Auth_MLSignerInfo_Certificate_CantFindCSCA`

CSCA-certificate for MLS-certificate digital signature verification is not found (see section 5.5.2). Marks all certificates contained in the checked master list.

- `ntfLDS_Auth_MLSignerInfo_Certificate_Revoked`

MLS-certificate revoked (see section 5.5.2). Marks all certificates contained in the checked master list.

- `ntfLDS_Auth_MLSignerInfo_Certificate_SignatureInvalid`

MLS-certificate signature verification failed (see sections 4.8.1). Marks all certificates contained in the checked master list.

6.4.44. eRFID_ErrorCodes

eRFID_ErrorCodes enumeration contains a set of error codes returned by the MCL functions or transferred to the user application by the callback-function.

```
enum eRFID_ErrorCodes
{
  RFID_Error_NoError                             = 0x00000001,
  RFID_Error_AlreadyDone                         = 0x00000002,
  RFID_Error_Failed                              = 0xffffffff,

  RFID_Error_NoChipDetected                      = 0x80010001,
  RFID_Error_NotAvailable                        = 0x80010002,

  RFID_Error_InvalidParameter                    = 0x80010004,
  RFID_Error_NotInitialized                      = 0x80010005,
  RFID_Error_NotEnoughMemory                     = 0x80010006,
  RFID_Error_InvalidDirectory                    = 0x80010008,
  RFID_Error_UnknownCommand                      = 0x80010009,
  RFID_Error_FileIOError                         = 0x8001000A,
  RFID_Error_Busy                                = 0x8001000B,
  RFID_Error_OldFirmware                         = 0x8001000C,
```

```
RFID_Error_PCSC_Failed                            = 0x80020000,
RFID_Error_PCSC_ReaderNotAvailable                = 0x80020001,
RFID_Error_PCSC_CantConnectCard                   = 0x80020002,
RFID_Error_PCSC_CardIsNotConnected                = 0x80020003,
RFID_Error_PCSC_OperationCancelled                = 0x80020004,
RFID_Error_PCSC_CardIsBusy                        = 0x80020005,
RFID_Error_PCSC_FailedSCard                       = 0x80020006,
RFID_Error_PCSC_ExtLe_Failed                      = 0x80020010,

RFID_LAYER6_SECURITY_MANAGER                      = 0x86000000,
RFID_LAYER6_APP_SELECTION_FAILURE                 = 0x86000001,

RFID_LAYER6_MUTUAL_AUTH_MAC_FAIL                  = 0x86000100,
RFID_LAYER6_MUTUAL_AUTH_ENC_FAIL                  = 0x86000101,
RFID_LAYER6_MUTUAL_AUTH_FAILURE                   = 0x86000102,

RFID_LAYER6_SM_DO8E_MISSING                       = 0x86000200,
RFID_LAYER6_SM_DO87_MISSING                       = 0x86000201,
RFID_LAYER6_SM_DO99_MISSING                       = 0x86000202,
RFID_LAYER6_SM_MAC_INCORRECT                      = 0x86000203,
RFID_LAYER6_SM_DO87_INCORRECT                     = 0x86000204,

RFID_LAYER6_NON_TLV_RESPONSE_DATA                 = 0x86000300,
RFID_LAYER6_WRONG_RND_ICC_LENGTH                  = 0x86000301,
RFID_LAYER6_INT_AUTH_FAILURE                      = 0x86000302,
FID_LAYER6_MSE_SET_KAT_FAILURE                    = 0x86000303,
RFID_LAYER6_MSE_SET_DST_FAILURE                   = 0x86000304,
RFID_LAYER6_PSO_CERTIFICATE_FAILURE               = 0x86000305,
RFID_LAYER6_MSE_SET_AT_FAILURE                    = 0x86000306,
RFID_LAYER6_GET_CHALLENGE_FAILURE                 = 0x86000307,
RFID_LAYER6_EXT_AUTH_FAILURE                      = 0x86000308,
RFID_LAYER6_GENERAL_AUTH_FAILURE                  = 0x86000309,

RFID_LAYER6_FILE_NOT_FOUND                        = 0x80006A82,
RFID_LAYER6_FILE_EOF1                             = 0x80006282,
RFID_LAYER6_FILE_EOF2                             = 0x80006B00,
RFID_LAYER6_INCORRECT_PARAMS                      = 0x80006A80,
RFID_LAYER6_NO_REFERENCE_DATA                     = 0x80006A88,
RFID_LAYER6_PWD_SUSPENDED                         = 0x800063C1,
RFID_LAYER6_PWD_BLOCKED                           = 0x800063C0,
RFID_LAYER6_PWD_DEACTIVATED                       = 0x80006283,
RFID_LAYER6_PWD_BLOCKED_2                         = 0x80006983,
RFID_LAYER6_PWD_DEACTIVATED_2                     = 0x80006984,
RFID_LAYER6_PWD_SUSPENDED_2                       = 0x80006985,
RFID_LAYER6_PWD_FAILED                            = 0x801063C0,

RFID_Error_NotPerformed                           = 0x83000000,
RFID_Error_Session_IsClosed                       = 0x83000001,
RFID_Error_Session_Terminal_UnsupportedOperation  = 0x83000002,

RFID_Error_Session_TerminalType_Unknown           = 0x83000010,
RFID_Error_Session_TerminalType_BadCertificate    = 0x83000011,
RFID_Error_Session_TerminalType_NotSet            = 0x83000012,

RFID_Error_Session_ProcedureType_Unknown          = 0x83000013,
RFID_Error_Session_ProcedureType_Unsupported      = 0x83000014,
RFID_Error_Session_ProcedureType_NotSet           = 0x83000015,
```

```
   RFID_Error_Session_AccessKey_UnknownType        = 0x83000016,
   RFID_Error_Session_AccessKey_UnsupportedSMType = 0x83000017,
   RFID_Error_Session_AccessKey_IncorrectSMType   = 0x83000018,
   RFID_Error_Session_AccessKey_Restricted        = 0x83000019,
   RFID_Error_Session_AccessKey_IncorrectData     = 0x8300001A,
   RFID_Error_Session_AccessKey_NotSet            = 0x8300001B,

   RFID_Error_Session_PwdManagement_NotAuthorized  = 0x8300001C,
   RFID_Error_Session_AccessControl_UnknownType   = 0x83000020,
   RFID_Error_Session_AccessControl_RequiresSM    = 0x83000021,
   RFID_Error_Session_AccessControl_RequiresPACE  = 0x83000022,
   RFID_Error_Session_AccessControl_RequiresCAKeys= 0x83000023,
   RFID_Error_Session_AccessControl_RequiresTA    = 0x83000024,
   RFID_Error_Session_AccessControl_RequiresCA    = 0x83000025,

   RFID_Error_Session_AccessControl_IncorrectOptionCA  = 0x83000026,

   RFID_Error_Session_AccessControl_CA_Failed     = 0x83000027,
   RFID_Error_Session_AccessControl_TA_Failed     = 0x83000028,
   RFID_Error_Session_AccessControl_AA_Failed     = 0x83000029,
   RFID_Error_Session_AccessControl_RI_Failed     = 0x8300002A,
   RFID_Error_Session_PA_SignatureCheckFailed     = 0x83000030,
   RFID_Error_Session_PA_HashCheckFailed          = 0x83000031,

   RFID_Error_Session_InvalidAuxData_DateOfExpiry  = 0x83000040,
   RFID_Error_Session_InvalidAuxData_DateOfBirth  = 0x83000041,
   RFID_Error_Session_InvalidAuxData_CommunityID  = 0x83000042,

   RFID_Error_Session_eSign_RequiresAppSelection  = 0x83000050,
   RFID_Error_Session_eSign_PIN_NotSet            = 0x83000051,
   RFID_Error_Session_eSign_PIN_NotVerified       = 0x83000052,

   RFID_Error_Session_IncorrectData               = 0x83000060,

   RFID_Error_Session_File_NotEnoughData          = 0x83010000,
   RFID_Error_Session_File_IncorrectData          = 0x83020000,
   RFID_Error_Session_File_UnexpectedData         = 0x83030000,
   RFID_Error_Session_File_Contents_UnexpectedData  = 0x83040000,
   RFID_Error_Session_File_WrongTag               = 0x83050000,
   RFID_Error_Session_File_CantUseData            = 0x83060000,
   RFID_Error_Session_File_CantReadData           = 0x83070000,
   RFID_Error_Session_File_AccessDenied           = 0x83080000,
};
```

The following code value:

- `RFID_Error_NoError`

Successful operation

- `RFID_Error_AlreadyDone`

Requested operation has already been performed

- `RFID_Error_Failed`

Error of operation execution (common case)

- `RFID_Error_NoChipDetected`

RFID-chip is absent in the scope of the reader antenna

- `RFID_Error_NotAvailable`

Requested operation unavailable

- `RFID_Error_InvalidParameter`

Incorrect command parameter

- `RFID_Error_NotInitialized`

SDK control library was not initialized

- `RFID_Error_NotEnoughMemory`

Insufficient memory for command execution

- `RFID_Error_InvalidDirectory`

Incorrect directory name

- `RFID_Error_UnknownCommand`

Unknown command

- `RFID_Error_FileIOError`

File input/output error

- `RFID_Error_Busy`

SDK control library is busy. Execution of the command is impossible

- `RFID_Error_OldFirmware`

It is required to update reader's firmware (see section 5.3)

- `RFID_Error_PCSC_Failed`

Error of command of data exchange with RFID-chip execution (common case)

- `RFID_Error_PCSC_ReaderNotAvailable`

RFID-chip reader unavailable

- `RFID_Error_PCSC_CantConnectCard`

Failed to connect with RFID-chip

- `RFID_Error_PCSC_CardIsNotConnected`

No active RFID-chip

- `RFID_Error_PCSC_OperationCancelled`

Data reading operation cancelled by the user

- `RFID_Error_PCSC_CardIsBusy`

Data exchange with RFID-chip takes place. Execution of the command is impossible

- `RFID_Error_PCSC_FailedSCard`

Error of `SCard` service when data exchanging with RFID-chip

- `RFID_Error_PCSC_ExtLe_Failed`

Error of executing command of extended length reading. Full reinitialization of RFID-chip is required (see section 5.4.4)

- `RFID_LAYER6_SECURITY_MANAGER`

Secure communication channel organization is required to access data (see sections 5.7.5, 5.8.3, 5.8.8)

- `RFID_LAYER6_APP_SELECTION_FAILURE`

Error of executing APDU-command of *Master File* or application selection [30]

- `RFID_LAYER6_MUTUAL_AUTH_MAC_FAIL`

Error of the cryptogram checksum validation during BAC procedure [2, section IV, § A5.3], [30]

- `RFID_LAYER6_MUTUAL_AUTH_ENC_FAIL`

Error of the cryptogram validation during BAC [2, section IV, § A5.3], [30]

- `RFID_LAYER6_MUTUAL_AUTH_FAILURE`

One of the errors of the APDU-commands:

1) `Mutual Authenticate` – performing the BAC [2, section IV, §A5.2];
2) `General Authenticate` – at the 4th step of PACE performance [24, part 3, §B1];
3) `General Authenticate` – performing the CA [24, part 3, §B2]

- `RFID_LAYER6_SM_DO8E_MISSING`

Absence of the checksum in the protected APDU-response from the RFID-chip (SM ‘`8E`’ data object) [2, section IV, §A5.3], [30]

- `RFID_LAYER6_SM_DO87_MISSING`

Absence of the data padding object in the protected APDU-response from the RFID-chip (SM ‘`87`’ data object) [2, section IV, §A5.3], [30]

- `RFID_LAYER6_SM_DO99_MISSING`

Absence of the command execution status in the protected APDU-response from the RFID-chip (SM ‘`99`’ data object) [2, section IV, §A5.3], [30]

- `RFID_LAYER6_SM_MAC_INCORRECT`

Error of the checksum validation of the protected APDU-response from the RFID-chip [2, section IV, §A5.3], [30]

- `RFID_LAYER6_SM_DO87_INCORRECT`

Incorrect contents of the data padding object in the protected APDU-response from the RFID-chip (SM ‘87’ data object) [2, section IV, §A5.3], [30]

- RFID_LAYER6_NON_TLV_RESPONSE_DATA

Response to Read Binary APDU-command, using 'B1' parameter for file data reading with an offset >32767, is not represented in the TLV-format [2, section III, § A.23], [30]

- RFID_LAYER6_GET_CHALLENGE_FAILURE

Error of Get Challenge APDU-command execution [30]

- RFID_LAYER6_WRONG_RND_ICC_LENGTH

Incorrect length of the data received as a result of Get Challenge APDU-command execution [30]

- RFID_LAYER6_INT_AUTH_FAILURE

Error of Internal Authenticate APDU-command execution when performing AA procedure [2, section IV, §A4.2], [30]

- RFID_LAYER6_MSE_SET_KAT_FAILURE

Error of MSE:Set KAT APDU-command execution when performing CA procedure, version 1 [1, §B.1.1], [24, part 3, §B.2], [30]

- RFID_LAYER6_MSE_SET_DST_FAILURE

Error of MSE:Set DST APDU-command execution when performing TA procedure [1, § B.2.1], [24, part 3, §B.3], [30]

- RFID_LAYER6_PSO_CERTIFICATE_FAILURE

Error of PSO:Verify Certificate APDU-command execution when performing TA procedure [1, §B.2.2], [24, part 3, §B.3], [30]

- RFID_LAYER6_MSE_SET_AT_FAILURE

Error of MSE:Set AT APDU-command execution [30] when performing the procedures:

1) PACE [24, part 3, §B.1];
2) TA [1, §B.2.3];
3) CA, version 2 [24, part 3, §B.2];
4) RI [24, part 3, §B.4];

- RFID_LAYER6_EXT_AUTH_FAILURE

Error of External Authenticate APDU-command execution when performing TA procedure [1, §B.2.5], [24, part 3, §B.3], [30]

- RFID_LAYER6_GENERAL_AUTH_FAILURE

Error of General Authenticate APDU-command execution [30] when performing PACE procedure at steps 1–3 [24, part 3, §B.1]

- RFID_LAYER6_FILE_NOT_FOUND

File not found. '6A 82' status code of APDU-command execution [30]

- RFID_LAYER6_FILE_EOF1

Attempt of reading outside the end of the file. '62 82' status code of APDU-command execution [30]

- RFID_LAYER6_FILE_EOF2

Attempt of reading outside the file end of the. '6B 00' status code of APDU-command execution [30]

- RFID_LAYER6_INCORRECT_PARAMS

Error of the execution of MSE:Set AT (variants of appearance – see RFID_LAYER6_MSE_SET_AT_FAILURE description) or General Authenticate (variants of appearance – see RFID_LAYER6_GENERAL_AUTH_FAILURE description).

Possible values in the context of command using:
1) the terminal does not support use of the selected type of password [24, part3, §B.11.1];
2) the selected algorithm is not supported [24, part3, §B.11.1];
3) initialization error [24, part3, §B.11.2];

- RFID_LAYER6_NO_REFERENCE_DATA

Unavailable data, pointed to by the APDU-command parameters

- RFID_LAYER6_PWD_SUSPENDED

Error of the execution of MSE:Set AT (when performing PACE) or General Authenticate (variants of appearance are analogue to RFID_LAYER6_GENERAL_AUTH_FAILURE). It means that the selected password is suspended. It is required to perform a procedure of password resuming (see sections 4.5, 5.8.19)

- RFID_LAYER6_PWD_BLOCKED

Error of the execution of MSE:Set AT (when performing PACE) or General Authenticate (variants of appearance are analogue to RFID_LAYER6_GENERAL_AUTH_FAILURE). It means that the selected password is blocked. It is required to perform a procedure of password unblocking (see sections 4.5, 5.8.19)

- RFID_LAYER6_PWD_DEACTIVATED

Error of the execution of MSE:Set AT (when performing PACE). It means that the selected password is deactivated. It is required to perform a procedure of password activation (see sections 4.5, 5.8.19)

- RFID_LAYER6_PWD_BLOCKED_2

Error of the execution of General Authenticate APDU-command or commands of work with *eSign* application [26]. It means that the selected password is blocked. It is required to perform procedure of password unblocking (see sections 4.5, 5.8.19)

- RFID_LAYER6_PWD_DEACTIVATED_2

Error of the execution of General Authenticate APDU-command when performing PACE procedure or commands of work with *eSign* application [26]. It means that the selected password is deactivated. It is required to perform procedure of password activation (see sections 4.5, 5.8.19)

- RFID_LAYER6_PWD_SUSPENDED_2

Error of General Authenticate APDU-command execution when performing PACE. It means that the selected password is suspended. It is required to perform procedure of password resuming (see sections 4.5, 5.8.19)

- RFID_LAYER6_PWD_FAILED

Error of the execution of MSE:Set AT when performing PACE procedure or General Authenticate (variants of appearance are analogue to RFID_LAYER6_GENERAL_AUTH_FAILURE).
It means that incorrect password value has been used. Low order 8 bits of code contain the remaining number of attempts for this password. The user application may try to repeat an attempt of performing the required procedure with other values (see sections 4.5, 5.8.19)

- RFID_Error_NotPerformed

Operation was not performed

- RFID_Error_Session_IsClosed

Session closed, operation impossible

- RFID_Error_Session_Terminal_UnsupportedOperation

Operation is not supported by the current type of terminal

- RFID_Error_Session_TerminalType_Unknown

Unknown type of terminal (see section 5.8.4)

- RFID_Error_Session_TerminalType_BadCertificate

Error of reading or analysis of the terminal certificate data (see section 5.8.4)

- RFID_Error_Session_TerminalType_NotSet

Terminal type was not defined for the current session (see section 5.8.4)

- RFID_Error_Session_ProcedureType_Unknown

Unknown authentication procedure type (see section 5.8.5)

- RFID_Error_Session_ProcedureType_Unsupported

Defined type terminal does not support this type of procedure (see section 5.8.5)

- RFID_Error_Session_ProcedureType_NotSet

Type of authentication procedure was not set for the current session (see section 5.8.5)

- RFID_Error_Session_AccessKey_UnknownType

Unknown access key type (see section 5.8.6)

- RFID_Error_Session_AccessKey_UnsupportedSMType

Unsupported type of secure data access procedure (see section 5.8.6)

- RFID_Error_Session_AccessKey_IncorrectSMType

Secure data access procedure does not allow to use the given key type (see section 5.8.6)

- RFID_Error_Session_AccessKey_Restricted

Key type is not supported by the current terminal type, or the rights to its use are insufficient

- RFID_Error_Session_AccessKey_IncorrectData

Incorrect key contents (empty or zero string)

- RFID_Error_Session_AccessKey_NotSet

Secure data access key was not set for the current session

- RFID_Error_Session_PwdManagement_NotAuthorized

Operation of password management is not authorized for the current terminal type

- RFID_Error_Session_AccessControl_UnknownType

Unknown type of the procedure of authentication or secure data access

- RFID_Error_Session_AccessControl_RequiresSM

Preliminary opening of the secure data access session is required (PACE or BAC)

- RFID_Error_Session_AccessControl_RequiresPACE

Preliminary opening of the secure data access session is required (PACE)

- RFID_Error_Session_AccessControl_RequiresCAKeys

Execution of TA preliminary step (for version 2) is required – computing CA ephemeral public keys (see sections 5.8.14)

- RFID_Error_Session_AccessControl_RequiresTA

Preliminary TA procedure is required

- RFID_Error_Session_AccessControl_RequiresCA

Preliminary CA procedure is required

- RFID_Error_Session_AccessControl_IncorrectOptionCA

Discrepancy between the selected CA variant on the preliminary and main stages (see section 5.8.14)

- RFID_Error_Session_AccessControl_CA_Failed

CA procedure failed

- RFID_Error_Session_AccessControl_TA_Failed

TA procedure failed

- RFID_Error_Session_AccessControl_AA_Failed

AA procedure failed

- `RFID_Error_Session_AccessControl_RI_Failed`

RI procedure failed

- `RFID_Error_Session_PA_SignatureCheckFailed`

Document security object digital signature verification failed (see section 5.8.12)

- `RFID_Error_Session_PA_HashCheckFailed`

Informational data group integrity verification failed (see section 5.8.13)

- `RFID_Error_Session_InvalidAuxData_DateOfExpiry`

Verification of auxiliary data (*date of expiry*) failed (see section 5.8.18)

- `RFID_Error_Session_InvalidAuxData_DateOfBirth`

Verification of auxiliary data (*age*) failed (see section 5.8.18)

- `RFID_Error_Session_InvalidAuxData_CommunityID`

Verification of auxiliary data (*`Community ID`*) failed (see section 5.8.18)

- `RFID_Error_Session_eSign_RequiresAppSelection`

Selection of *`eSign`* application is required to access its functionality (see section 5.8.20)

- `RFID_Error_Session_eSign_PIN_NotSet`

It is required to set the value of *eSign-PIN* for the current session (see section 5.8.20)

- `RFID_Error_Session_eSign_PIN_NotVerified`

It is required to execute verification of *eSign-PIN* for the current session (see section 5.8.20)

- `RFID_Error_Session_IncorrectData`

Incorrect session object data (see section 5.8.22)

- `RFID_Error_Session_File_NotEnoughData`

No sufficient data for creation of the file contents ASN.1 object. Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_IncorrectData`

Incorrect data of the file contents ASN.1 object. Low orer `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_UnexpectedData`

Incompliance of the structure of the file contents ASN.1 object with the structure given in the respective specification. Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_Contents_UnexpectedData`

Incompliance of the structure of the formed ASN.1-objects with the requirements of specification (in the context of specific file). Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_WrongTag`

Incorrect value of the data group tag. Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_CantUseData`

Use of the read data is impossible (for example, when detecting any inconsistances in `dplStrictISO` mode, see section 5.2). Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_CantReadData`

Error of physical data reading. Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

- `RFID_Error_Session_File_AccessDenied`

Error of access to the protected data groups. Low order `WORD` of the code contains the file identifier (one of **`eRFID_DataFile_Type`** values)

6.4.45. eRFID_ControlRF

`eRFID_ControlRF` enumeration contains a set of values for use for SDK main control library initialization (see sections 5.4.1, 5.4.2).

```
enum eRFID_ControlRF
{
  CtrlRF_Auto         = 0,
  CtrlRF_Manual       = 0x00000001,
  CtrlRF_14443_4_Only = 0x00000040,
};
```

Mode constant values:

`CtrlRF_Auto`	–	automatic detection of RFID-chip presence in the scope of the reader antenna;
`CtrlRF_Manual`	–	manual detection of RFID-chip presence in the scope of the reader antenna;
`CtrlRF_14443_4_Only`	–	ignoring the RFID-chips with support of only ISO/IEC 14443-3 protocol (MIFARE® Classic Protocol).

In the *automatic mode* of detection, when the chip appears in the scope of the reader antenna, a **`RFID_Notification_DocumentReady`** message will be generated automatically (see section 4.4) and in the future, after termination of data reading operation from the memory of the current chip, search of a new chip will be renewed.

In the *manual mode* search of chip presence in the scope of the reader antenna is performed (resumed) only if executing **`RFID_Command_DocumentDone`** command and is single. **`RFID_Notification_DocumentReady`** message is generated only if a chip presence status has changed since the previous execution of this command with such a single search.

Thus, the operation of RFID-chip detection (reading) in the manual mode should always be initiated by executing **RFID_Command_DocumentDone** command.

6.4.46. eDataProcessingLevel

eDataProcessingLevel enumeration contains a set of constants that define the level of SDK reaction to detection of any inconsistencies to specifications when analyzing the data or performing any operations (see section 5.2).

```
enum eDataProcessingLevel
{
   dplMaxAvailable  = 0,
   dplStrictISO     = 1,
};
```

Level constant values:

`dplMaxAvailable` – the least strict;
`dplStrictISO` – the most strict.

6.4.47. eRFID_AuthenticationProcedureType

eRFID_AuthenticationProcedureType enumeration contains a set of constants that define the type of performed procedure of document authentication within the context of the communication session with electronic document (see section 5.8.5).

```
enum eRFID_AuthenticationProcedureType
{
   aptUndefined  = 0,
   aptStandard   = 1,
   aptAdvanced   = 2,
   aptGeneral    = 3,
};
```

Value of procedure type constants:

`aptUndefined` – not defined;
`aptStandard` – standard;
`aptAdvanced` – advanced;
`aptGeneral` – general authentication procedure.

6.4.48. eRFID_Password_Type

eRFID_Password_Type enumeration contains a set of constants that define the type of key to access the protected data (see section 6.3.82).

```
enum eRFID_Password_Type
{
   pptUnknown     = 0,
   pptMRZ         = 1,
   pptCAN         = 2,
   pptPIN         = 3,
   pptPUK         = 4,
   pptPIN_eSign   = 5,
   pptSAI         = 6,
};
```

Value of constants:

pptUnknown	–	unknown type;
pptMRZ	–	MRZ;
pptCAN	–	CAN;
pptPIN	–	PIN;
pptPUK	–	PUK;
pptPIN_eSign	–	*eSign-PIN*;
pptSAI	–	Scanning Area Identifier (for *eDL* application).

6.4.49. eRFID_TerminalType

eRFID_TerminalType enumeration contains a set of constants that define the type of terminal within the context of the communication session with electronic document (see section 6.3.83).

```
enum eRFID_TerminalType
{
   tetUndefined                = 0,
   tetInspectionSystem         = 1,
   tetAuthenticationTerminal   = 2,
   tetSignatureTerminal        = 3,
   tetUnauthenticatedTerminal  = 4,
};
```

Value of constants of terminal types:

tetUndefined	–	not defined;
tetInspectionSystem	–	inspection system;
tetAuthenticationTerminal	–	authentication terminal;
tetSignatureTerminal	–	signature terminal;
tetUnauthenticatedTerminal	–	unauthenticated terminal.

6.4.50. eRFID_TerminalAuthorizationRequirement

eRFID_TerminalAuthorizationRequirement enumeration contains a set of constants used in setting the combination of access rights to information and functional capabilities requested from the RFID-chip (or delegated by it) (see section 6.3.83).

```
enum eRFID_TerminalAuthorizationRequirement
{
   tar_IS_ePassport_DG3                = 0x00000001,
   tar_IS_ePassport_DG4                = 0x00000002,
   tar_IS_ePassport_AllDG              = 0x00000003,

   tar_AT_eID_Read_DG1                 = 0x00000001,
   tar_AT_eID_Read_DG2                 = 0x00000002,
   tar_AT_eID_Read_DG3                 = 0x00000004,
   tar_AT_eID_Read_DG4                 = 0x00000008,
   tar_AT_eID_Read_DG5                 = 0x00000010,
   tar_AT_eID_Read_DG6                 = 0x00000020,
   tar_AT_eID_Read_DG7                 = 0x00000040,
   tar_AT_eID_Read_DG8                 = 0x00000080,
   tar_AT_eID_Read_DG9                 = 0x00000100,
   tar_AT_eID_Read_DG10                = 0x00000200,
   tar_AT_eID_Read_DG11                = 0x00000400,
   tar_AT_eID_Read_DG12                = 0x00000800,
   tar_AT_eID_Read_DG13                = 0x00001000,
   tar_AT_eID_Read_DG14                = 0x00002000,
   tar_AT_eID_Read_DG15                = 0x00004000,
   tar_AT_eID_Read_DG16                = 0x00008000,
   tar_AT_eID_Read_DG17                = 0x00010000,
   tar_AT_eID_Read_DG18                = 0x00020000,
   tar_AT_eID_Read_DG19                = 0x00040000,
   tar_AT_eID_Read_DG20                = 0x00080000,
   tar_AT_eID_Read_DG21                = 0x00100000,
   tar_AT_eID_Read_AllDG               = 0x001FFFFF,

   tar_AT_eID_Write_DG17               = 0x00200000,
   tar_AT_eID_Write_DG18               = 0x00400000,
   tar_AT_eID_Write_DG19               = 0x00800000,
   tar_AT_eID_Write_DG20               = 0x01000000,
   tar_AT_eID_Write_DG21               = 0x02000000,
   tar_AT_eID_Write_AllDG              = 0x03E00000,

   tar_AT_Func_InstallQCert            = 0x00000001,
   tar_AT_Func_InstallCert             = 0x00000002,
   tar_AT_Func_PINManagement           = 0x00000004,
   tar_AT_Func_CAN_Allowed             = 0x00000008,
   tar_AT_Func_PrivilegedTerminal      = 0x00000010,
   tar_AT_Func_RestrictedIdent         = 0x00000020,
   tar_AT_Func_Verify_CommunityID      = 0x00000040,
   tar_AT_Func_Verify_Age              = 0x00000080,
   tar_AT_Func_Full                    = 0x000000FF,

   tar_ST_Gen_QualifiedSignature       = 0x00000001,
   tar_ST_Gen_Signature                = 0x00000002,
   tar_ST_Gen_Full                     = 0x00000003,
};
```

The value of the constants corresponds to the individual flags of a combination of access rights to protected data and functional capabilities of the electronic document from the terminal certificates for terminals of different types (see section 4.7).

6.4.51. eRFID_FileID_Type

eRFID_FileID_Type enumeration contains a set of constants that define the type of file identifier and its addressing (selection) method (see section 6.3.87).

```
enum eRFID_FileID_Type
{
   fidtUndefined    = 0,
   fidtMF_FullName  = 1,
   fidtMF_ShortName = 2,
   fidtDF_FullName  = 3,
   fidtDF_ShortName = 4,
   fidtLocal_Path   = 5,
};
```

Value of constants of types of file identifier:

`fidtUndefined`	–	not defined;
`fidtMF_FullName`	–	full, with *Master File* prefix (`'3F 00'`);
`fidtMF_ShortName`	–	short, relative to *Master File*;
`fidtDF_FullName`	–	fill, relative to the current application;
`fidtDF_ShortName`	–	short, addressing relative to the current application;
`fidtLocal_Path`	–	full, relative to the current application, file data reading is not performed.

6.4.52. eRFID_AccessControl_ProcedureType

eRFID_AccessControl_ProcedureType enumeration contains a set of constants that define the type of authentication or secure data access procedure (see section 6.3.90).

```
enum eRFID_AccessControl_ProcedureType
{
   acptUndefined = 0,
   acptBAC       = 1,
   acptPACE      = 2,
   acptCA        = 3,
   acptTA        = 4,
   cptAA         = 5,
   acptRI        = 6,
};
```

Value of procedure type constants:

`acptUndefined`	–	type is not defined;

`acptBAC` – BAC/BAP;
`acptPACE` – PACE;
`acptCA` – CA;
`acptTA` – TA;
`acptAA` – AA;
`acptRI` – RI.

6.4.53. eRFID_TerminalAuthenticationType

eRFID_TerminalAuthenticationType enumeration contains a set of constants that define the order of terminal authentication procedure (see section 6.3.95).

```
enum eRFID_TerminalAuthenticationType
{
   tatDefault    = 0,
   tatOnline     = 1,
   tatStepByStep = 2,
};
```

Value of mode constants:

`tatDefault` – automatic, by default;
`tatOnline` – step-by-step, Online-authentication;
`tatStepByStep` – step-by-step interruptible.

6.4.54. eRFID_AuxiliaryDataType

eRFID_AuxiliaryDataType enumeration contains a set of constants that define the type of verified auxiliary data (see section 5.8.18).

```
enum eRFID_AuxiliaryDataType
{
   adtAge           = 1,
   adtDateOfExpiry  = 2,
   adtCommunityID   = 3,
};
```

Constant values:

`adtAge` – DO's age;
`adtDateOfExpiry` – date of expiry;
`adtCommunityID` – Community ID.

6.4.55. eRFID_SectorKeyType

eRFID_SectorKeyType enumeration contains a set of constants that define the type of terminal sector key when performing RI (see section 5.8.17).

```
enum eRFID_SectorKeyType
```

```
{
   spkiSectorKey1 = 1,
   spkiSectorKey2 = 2,
};
```

Constant values:

`spkiSectorKey1` – terminal sector key 1;
`spkiSectorKey2` – terminal sector key 2.

6.4.56. eRFID_Application_Type

`eRFID_Application_Type` enumeration contains a set of constants that define the type of application within the context of the communication session with electronic document (see section 6.3.67).

```
enum eRFID_Application_Type
{
   at_Unspecified= 0,
   at_ePassport   = 1,
   at_eID         = 2,
   at_eSign       = 3,
   at_eDL         = 4,
   at_RootFiles   = at_Unspecified,
};
```

Value of constants:

`at_Unspecified` – not defined;
`at_ePassport` – *`ePassport`* application;
`at_eID` – *`eID`* application;
`at_eSign` – *`eSign`* application;
`at_eDL` – *`eDL`* application;
`at_RootFiles` – *Master File*.

6.4.57. eRFID_DataFile_Type

`eRFID_DataFile_Type` ynumeration contains a set of constants that define the file type (or logical belonging of the data object) within the context of the communication session with electronic document (see sections 5.7.3, 5.8.8, 5.8.11, 5.8.21, 6.3.68).

```
enum eRFID_DataFile_Type
{
   dftUnspecified          = 0,

   dftPassport_DG1         = 1,
   dftPassport_DG2         = 2,
   dftPassport_DG3         = 3,
   dftPassport_DG4         = 4,
   dftPassport_DG5         = 5,
```

```
    dftPassport_DG6         = 6,
    dftPassport_DG7         = 7,
    dftPassport_DG8         = 8,
    dftPassport_DG9         = 9,
    dftPassport_DG10        = 10,
    dftPassport_DG11        = 11,
    dftPassport_DG12        = 12,
    dftPassport_DG13        = 13,
    dftPassport_DG14        = 14,
    dftPassport_DG15        = 15,
    dftPassport_DG16        = 16,
    dftPassport_DG17        = 17,
    dftPassport_DG18        = 18,
    dftPassport_DG19        = 19,
    dftPassport_DG20        = 20,
    dftPassport_SOD         = 21,
    dftPassport_CVCA        = 22,
    dftPassport_COM         = 23,

    dftID_DG1               = 101,
    dftID_DG2               = 102,
    dftID_DG3               = 103,
    dftID_DG4               = 104,
    dftID_DG5               = 105,
    dftID_DG6               = 106,
    dftID_DG7               = 107,
    dftID_DG8               = 108,
    dftID_DG9               = 109,
    dftID_DG10              = 110,
    dftID_DG11              = 111,
    dftID_DG12              = 112,
    dftID_DG13              = 113,
    dftID_DG14              = 114,
    dftID_DG15              = 115,
    dftID_DG16              = 116,
    dftID_DG17              = 117,
    dftID_DG18              = 118,
    dftID_DG19              = 119,
    dftID_DG20              = 120,
    dftID_DG21              = 121,

    dftDL_COM               = 150,
    dftDL_DG1               = 151,
    dftDL_DG2               = 152,
    dftDL_DG3               = 153,
    dftDL_DG4               = 154,
    dftDL_DG5               = 155,
    dftDL_DG6               = 156,
    dftDL_DG7               = 157,
    dftDL_DG8               = 158,
    dftDL_DG9               = 159,
    dftDL_DG10              = 160,
    dftDL_DG11              = 161,
    dftDL_DG12              = 162,
    dftDL_DG13              = 163,
    dftDL_DG14              = 164,
    dftDL_SOD               = 165,
    dftDL_CE                = 166,
```

```
   dftDL_CVCA              = 167,

   dftPACE_CardAccess      = 200,
   dftPACE_CardSecurity    = 201,
   dftPACE_ChipSecurity    = 202,

   dftMIFARE_Data          = 300,
   dftMIFARE_Validity      = 301,

   dftAuthenticityV2       = 302,
   dftATR                  = 400,
   dft_eSign_PK            = 500,
   dft_eSign_SignedData    = 501,
   dftCertificate          = 600,
   dftMasterList           = 601,
   dftDefectList           = 602,
   dftApp_Directory        = 700,
   dftSession              = 701,
   dftLogData              = 702,
   dftChipProperties       = 703,
   dftUserDefined          = 1000,
};
```

6.4.58. eRFID_CertificateOrigin

eRFID_CertificateOrigin enumeration contains a set of constants that define the source of certificate used in the procedure of document security object digital signature verification (see section 6.3.73).

```
enum eRFID_CertificateOrigin
{
  coUndefined         = 0,
  coPKD               = 1,
  coSecurityObject    = 2,
  coUserDefined       = 3,
  coMasterList_PKD    = 4,
  coMasterList_SO     = 5,
  coDefectList_SO     = 6,
  coDeviationList_SO  = 7,
  coBlackList_SO      = 8,
};
```

Constant values:

coUndefined	–	the source is not defined;
coPKD	–	local PKD;
coSecurityObject	–	document security object;
coUserDefined	–	user-defined;
coMasterList_PKD	–	contents of the Master List;
coMasterList_SO	–	security object of the Master List.
coDefectList_SO	–	security object of the Defect List,

`coDeviationList_SO` – security object of the Deviation List,
`coBlackList_SO` – security object of the Black List.

6.4.59. eRFID_CertificateType

`eRFID_CertificateType` enumeration contains a set of constants that define the type of certificate used in the procedure of document security object digital signature verification (see section 6.3.73).

```
enum eRFID_CertificateType
{
  ctUndefined  = 0,
  ctCSCA       = 1,
  ctCSCALink   = 2,
  ctDS         = 3,
  ctMLS        = 4,
  ctDevLS      = 5,
  ctDefLS      = 6,
};
```

Constant values:

`ctUndefined` – type is not defined;
`ctCSCA` – CSCA;
`ctCSCALink` – CSCA-link;
`ctDS` – DS;
`ctMLS` – Master List signer,
`ctDevLS` – Deviaton List signer,
`ctDefLS` – Defect List signer.

6.4.60. eRFID_PasswordManagementAction

Перечисление **`eRFID_PasswordManagementAction`** enumeration contains a set of constants that define the type of conducting operation with the secure data access key in the scenario operation mode (see sections 4.5, 5.9.4.3).

```
enum eRFID_PasswordManagementAction
{
  pmaUndefined           = 0,
  pmaChangeCAN           = 1,
  pmaChangePIN           = 2,
  pmaActivatePIN         = 3,
  pmaDeactivatePIN       = 4,
  pmaUnblockPIN          = 5,
  pmaResumePIN           = 6,
  pmaUnblock_eSignPIN    = 7,
  pmaCreate_eSignPIN     = 8,
  pmaTerminate_eSignPIN  = 9,
  pmaChange_eSignPIN     = 10,
```

```
};
```

Constant values:

`pmaUndefined`	– type is not defined;
`pmaChangeCAN`	– changing CAN;
`pmaChangePIN`	– changing PIN;
`pmaActivatePIN`	– activating PIN;
`pmaDeactivatePIN`	– deactivating PIN;
`pmaUnblockPIN`	– unblocking PIN;
`pmaResumePIN`	– resuming PIN;
`pmaUnblock_eSignPIN`	– unblocking eSign-PIN;
`pmaCreate_eSignPIN`	– creating eSign-PIN;
`pmaTerminate_eSignPIN`	– terminating eSign-PIN;
`pmaChange_eSignPIN`	– changing eSign-PIN.

6.4.61. eRFID_PasswordPostDialogAction

eRFID_PasswordPostDialogAction enumeration contains a set of constants that define an action action to be taken after the closing the dialog window of secure data access key management in the scenario operation mode (see section 5.9.4.4).

```
enum eRFID_PasswordPostDialogAction
{
  ppaUndefined       = 0,
  ppaRetry           = 1,
  ppaChangeType      = 2,
  ppaResume          = 3,
  ppaUnblock         = 4,
  ppaActivate        = 5,
  ppaDeactivate      = 6,
  ppaChange          = 7,
};
```

Constant values:

`ppaUndefined`	– action is not defined;
`ppaRetry`	– use the new value of the current key type;
`ppaChangeType`	– change the type of the key;
`ppaResume`	– conduct the operation of key resuming (for PIN);
`ppaUnblock`	– conduct the operation of key unblocking (for PIN);
`ppaActivate`	– conduct the operation of key activation (for PIN);
`ppaDeactivate`	– conduct the operation of key deactivation (for PIN);
`ppaChange`	– conduct the operation of key changing (for CAN, PIN).

6.4.62. eRFID_TerminalAuthenticationToSignDataType

eRFID_TerminalAuthenticationToSignDataType enumeration contains a set of constants that define type of data transmitted to the user application on the second step of TA procedure in *Online* and step mode (see section 5.8.15) or in the scenario operation mode (see section 5.9.4.7)

```
enum eRFID_TerminalAuthenticationToSignDataType
{
  tatsdtPlainData    = 0,
  tatsdtHashValue    = 1,
};
```

Constant values:

`tatsdtPlainData` –	data for signature generation transmitted;
`tatsdtHashValue` –	hash value of the data for signature generation transmitted.

6.5. SDK COMMAND SYSTEM (ERFID_COMMANDS)

The main library function, by which the user application can initiate all necessary actions for work with RFID-chips, is **`_RFID_ExecuteCommand()`** function (see section 6.1.4). It takes a command triplet as the parameters: *command code* (`command` parameter), *command input parameter* (`params` parameter) and a *pointer to the container* (`result` parameter) for return results of the command.

> **ATTENTION!** In some cases, the purpose of the parameters of **`_RFID_ExecuteCommand()`** function may vary.

`eRFID_Commands` enumeration contains a set of command codes supported by the current version of the SDK control library.

```
enum eRFID_Commands
{
   RFID_Command_Get_AvailableGraphicFormats        = 0x00000013,

   RFID_Command_Get_DeviceCount                    = 0x00000001,
   RFID_Command_Get_CurrentDevice                  = 0x00000002,
   RFID_Command_Set_CurrentDevice                  = 0x00000003,

   RFID_Command_Get_DeviceFirmwareVersion          = 0x00000007,
   RFID_Command_Get_DeviceDescription              = 0x00000016,

   RFID_Command_Get_DeviceDriverVersion            = 0x00000017,
   RFID_Command_Get_DeviceInstanceID               = 0x00000100,
   RFID_Command_Get_ParentInstanceID               = 0x00000101,
   RFID_Command_Get_DeviceHardwareID               = 0x00000102,
   RFID_Command_Get_CodeTranscription              = 0x00000103,
   RFID_Command_SelectDeviceByName                 = 0x00000040,
   RFID_Command_SelectDeviceBySN                   = 0x00000041,
   RFID_Command_Get_DeviceSN                       = 0x00000042,

   RFID_Command_BuildLog                           = 0x00000019,
   RFID_Command_FlushLog                           = 0x00000020,
   RFID_Command_LogDirectory                       = 0x00000021,
   RFID_Command_UseDeviceDriverLog                 = 0x00000022,

   RFID_Command_Set_CheckResultHeight              = 0x0000002E,

   RFID_Command_SetCryptKey                        = 0x00000008,
   RFID_Command_GetCryptKey                        = 0x00000009,

   RFID_Command_SetMIFARE_KeyMode                  = 0x0000000F,
   RFID_Command_GetMIFARE_KeyMode                  = 0x00000010,

   RFID_Command_SetMIFARE_KeyTable                 = 0x00000011,
   RFID_Command_GetMIFARE_KeyTable                 = 0x00000012,

   RFID_Command_Get_OperationalBaudRate            = 0x00000014,
   RFID_Command_Set_OperationalBaudRate            = 0x00000015,
```

```
RFID_Command_Set_PassivePKD                        = 0x00000032,
RFID_Command_Get_PassivePKD                        = 0x00000033,
RFID_Command_Set_EAC_PKD                           = 0x00000034,
RFID_Command_Get_EAC_PKD                           = 0x00000035,

RFID_Command_ReadCardProperties                    = 0x00000004,
RFID_Command_ReadCardPropertiesExt                 = 0x00000069,
RFID_Command_ReadCardPropertiesExt2                = 0x00000080,
RFID_Command_ReadProtocol4                         = 0x00000005,
RFID_Command_ReadProtocol3                         = 0x00000006,
RFID_Command_CancelReading                         = 0x0000001A,

RFID_Command_DocumentDone                          = 0x0000000A,
RFID_Command_IsDocument                            = 0x0000000B,
RFID_Command_ParseRawData                          = 0x00000018,
RFID_Command_ClearResults                          = 0x0000001B,
RFID_Command_Set_DetectionMode                     = 0x00000081,

RFID_Command_SetDataProcessingLevel                = 0x00000050,
RFID_Command_GetDataProcessingLevel                = 0x00000051,

RFID_Command_SetTransferBufferSize                 = 0x00000054,
RFID_Command_GetTransferBufferSize                 = 0x00000055,

RFID_Command_SetUserDefinedFilesToRead             = 0x00000060,

RFID_Command_Set_DS_Cert_Priority                  = 0x00000062,
RFID_Command_Get_DS_Cert_Priority                  = 0x00000063,

RFID_Command_Set_TrustedPKD                        = 0x0000006B,
RFID_Command_Get_TrustedPKD                        = 0x0000006A,

RFID_Command_Set_ProfilerType                      = 0x00000070,
RFID_Command_Get_ProfilerType                      = 0x00000071,

RFID_Command_Set_DefaultPACEOption                 = 0x00000072,
RFID_Command_Get_DefaultPACEOption                 = 0x00000073,

RFID_Command_Set_OnlineTAToSignDataType            = 0x00000074,
RFID_Command_Get_OnlineTAToSignDataType            = 0x00000075,

RFID_Command_Set_Processing_Amendment              = 0x00000078,
RFID_Command_Set_ParsedCustomDataType              = 0x00000079,

RFID_Command_Set_UseExternalCSCA                   = 0x0000007A,
RFID_Command_Get_UseExternalCSCA                   = 0x0000007B,

RFID_Command_Set_Graphics_CompressionRatio         = 0x0000007E,
RFID_Command_Get_Graphics_CompressionRatio         = 0x0000007F,
RFID_Command_Session_Open                          = 0x00001000,
RFID_Command_Session_SelectApplication             = 0x00001001,
RFID_Command_Session_AccessControlProc             = 0x00001002,
RFID_Command_Session_ReadFile                      = 0x00001003,
RFID_Command_Session_PA_CheckSO                    = 0x00001004,
RFID_Command_Session_PA_CheckFile                  = 0x00001005,
RFID_Command_Session_Close                         = 0x00001006,
RFID_Command_Session_ReadMifare                    = 0x00001007,
RFID_Command_Session_SetAccessKey                  = 0x00001008,
```

```
   RFID_Command_Session_SetTerminalType            = 0x00001009,
   RFID_Command_Session_SetProcedureType           = 0x0000100A,
   RFID_Command_Session_WriteFile                  = 0x0000100B,
   RFID_Command_Session_Verify                     = 0x0000100C,

   RFID_Command_Session_Password_ChangePIN         = 0x0000100D,
   RFID_Command_Session_Password_ChangeCAN         = 0x0000100E,
   RFID_Command_Session_Password_UnblockPIN        = 0x0000100F,
   RFID_Command_Session_Password_ActivatePIN       = 0x00001010,
   RFID_Command_Session_Password_DeactivatePIN     = 0x00001011,

   RFID_Command_Session_PA_IsFileCheckAvailable    = 0x00001012,

   RFID_Command_Session_eSign_CreatePIN            = 0x00001020,
   RFID_Command_Session_eSign_ChangePIN            = 0x00001021,

   RFID_Command_Session_eSign_UnblockPIN           = 0x00001022,
   RFID_Command_Session_eSign_TerminatePIN         = 0x00001023,
   RFID_Command_Session_eSign_VerifyPIN            = 0x00001024,
   RFID_Command_Session_eSign_GenerateKeyPair      = 0x00001025,
   RFID_Command_Session_eSign_TerminateKeyPair     = 0x00001026,
   RFID_Command_Session_eSign_SignData             = 0x00001027,

   RFID_Command_Session_LoadData                   = 0x00001030,
   RFID_Command_Session_SaveData                   = 0x00001031,
   RFID_Command_Session_LoadData_Reparse           = 0x00001032,

   RFID_Command_Scenario_Process                   = 0x00003000,
   RFID_Command_Set_TCC_Params                     = 0x00005000,
};
```

The description of each command is given below as follows:

- command code
- input parameter `params`
- output parameter `result`
- command assignment
- short description

The terms of *«input»/«output»* for the function parameter speak about its use either as an input parameter of the command, or it serves for receiving the data generated during the process of command execution.

6.5.1. RFID_Command_Get_AvailableGraphicFormats

Input parameter:	not used
Output parameter:	`char **`
Assignment:	acquisition of the list of graphic file format extensions, available for use when storing graphic data (see sections 5.6.3, 6.1.6)

This command initializes the pointer located at the address in `result` parameter by the pointer to the string with the current combination of formats.

6.5.2. RFID_Command_Get_DeviceCount

Input parameter: not used
Output parameter: `long *`
Assignment: determination of the total number of RFID-chip readers actually connected to the PC (see section 5.3)

6.5.3. RFID_Command_Get_CurrentDevice

Input parameter: not used
Output parameter: `long *`
Assignment: determination of the index of the current active RFID-chip reader (see section 5.3)

6.5.4. RFID_Command_Set_CurrentDevice

Input parameter: `long`
Output parameter: not used
Assignment: activation of the reader with the given index from the general list (see section 4.3)

Device index in the general list is given in `params` parameter.

6.5.5. RFID_Command_Get_DeviceFirmwareVersion

Input parameter: `long`
Output parameter: `long *`
Assignment: determination of the firmware version of the RFID-chip reader (see sections 5.3, 6.3.66)

Device index in the general list is given in `params` parameter.

Version of reader firmware is represented in `'A.B'` format, where

```
A = HIBYTE(LOWORD())
B = LOBYTE(LOWORD())
```

6.5.6. RFID_Command_Get_DeviceDescription

Input parameter: `long`
Output parameter: `char **`
Assignment: acquisition of the symbolic name of the reader from the general list (see section 5.3)

Device index in the general list is given in `params` parameter.

This command initializes the pointer at the address in `result` parameter with the pointer to the string with a symbolic reader name.

6.5.7. RFID_Command_Get_DeviceDriverVersion

Input parameter:	`long`
Output parameter:	`long *`
Assignment:	determination of RFID-chip reader driver version (see sections 5.3, 6.3.66)

Device index in the general list is given in `params` parameter.

Version of reader driver is represented in '`A.B.C.D`' format, where

```
A = HIBYTE(HIWORD())
B = LOBYTE(HIWORD())
C = HIBYTE(LOWORD())
D = LOBYTE(LOWORD())
```

6.5.8. RFID_Command_Get_DeviceInstanceID

Input parameter:	`long`
Output parameter:	`char **`
Assignment:	acquisition of the symbolic system identifier of the reader device instance, determined by Windows API `SetupDiGetDeviceInstanceID()` function (see section 5.3)

Device index in the general list is given in `LOWORD(params)`.

This command initializes the pointer located at the address in `result` parameter, with the pointer to the string with identifier.

Example of identifier value: `USB\VID_1C6A&PID_7051\6&13F14847&0&3`

6.5.9. RFID_Command_Get_ParentInstanceID

Input parameter:	`long`
Output parameter:	`char **`
Assignment:	acquisition of the symbolic system identifier of the device instance, to which the RFID-chip is physically connected (in most cases it is USB Hub), determined by Windows API `CM_Get_Device_ID()` function (see section 5.3)

Device index in the general list is given in `LOWORD(params)`, `HIWORD(params)` – index of parent device in the tree of mutual connections. Connection tree formed directly from the parent device and ends at the root system resource.

This command initializes the pointer located at the address in `result` parameter, with the pointer to the string with identifier.

Example of identifier value: `USB\VID_05E3&PID_0608\5&2CA10F73&0&2`

6.5.10. RFID_Command_Get_DeviceHardwareID

Input parameter:	`long`
Output parameter:	`char **`
Assignment:	acquisition of the symbolic system identifier of RFID-chip reader, determined by Windows API `SetupDiGetDeviceRegistryProperty()` function (see section 5.3)

Device index in the general list is given in `LOWORD(params)`.

This command initializes the pointer located at the address in `result` parameter, with the pointer to the string with identifier.

Example of identifier value: `USB\VID_1C6A&PID_7051&REV_0000`

6.5.11. RFID_Command_Get_CodeTranscription

Input parameter:	`long`
Output parameter:	`char **`
Assignment:	acquisition of the abbreviation of a notification or SDK function return code (see section 5.2)

Numerical code of event/status is specified in `params` parameter.

This command initializes the pointer located at the address in `result` parameter, with the pointer to the character string.

6.5.12. RFID_Command_SelectDeviceByName

Input parameter:	`char *`
Output parameter:	not used
Assignment:	RFID-chip reader activation by the symbolic string of the system UID of the parent Hub (see section 5.3)

6.5.13. RFID_Command_SelectDeviceBySN

Input parameter:	`long`
Output parameter:	not used
Assignment:	reader activation by the serial number of the RFID-chip reader (see section 5.3)

6.5.14. RFID_Command_Get_DeviceSN

Input parameter:	`long`
Output parameter:	`long *`
Assignment:	determination of the serial number of the RFID-chip reader (see section 5.3)

Device index in the general list is given in `params` parameter.

6.5.15. RFID_Command_BuildLog

Input parameter:	`bool`
Output parameter:	not used
Assignment:	activation/deactivation of SDK logging (see section 5.5.1)

6.5.16. RFID_Command_FlushLog

Input parameter:	`char *`
Output parameter:	not used
Assignment:	recording of the current file of SDK log under the assigned file name (see section 5.5.1)

Full log file name in UTF8 format is given in `params` parameter.

6.5.17. RFID_Command_LogDirectory

Input parameter:	`char *`
Output parameter:	not used
Assignment:	definition of the directory of SDK log file recording (see section 5.5.1)

Full directory name (in UTF8 format) of SDK log file recording is given in `params` parameter.

6.5.18. RFID_Command_Set_CheckResultHeight

Input parameter:	`long`
Output parameter:	not used

Assignment:	setting the required height of images (in pixels) requested using **`_RFID_CheckResultFromList()`** function (see section 6.1.6)

6.5.19. RFID_Command_SetCryptKey

Input parameter:	`char *`
Output parameter:	not used
Assignment:	setting the secure data access key (MRZ) when working in the batch mode (see section 5.7.5)

6.5.20. RFID_Command_GetCryptKey

Input parameter:	not used
Output parameter:	`char **`
Assignment:	acquisition of the current value of the secure data access key (MRZ) when working in the batch mode (see section 5.7.5)

This command initializes the pointer located at the address in `result` parameter, with the pointer to the string with the current key value.

6.5.21. RFID_Command_SetMIFARE_KeyMode

Input parameter:	`long`
Output parameter:	not used
Assignment:	setting the authentication mode type for data reading via MIFARE® Classic Protocol (see section 5.7.3)

One of the constants **`eMIFARE_KeyMode`** is given in the parameter `params`.

6.5.22. RFID_Command_GetMIFARE_KeyMode

Input parameter:	not used
Output parameter:	`long *`
Assignment:	acquisition of the current value of the authentication mode type for data reading via MIFARE® Classic Protocol (see section 5.7.3)

6.5.23. RFID_Command_SetMIFARE_KeyTable

Input parameter:	`TMIFARE_KeyTable *`
Output parameter:	not used
Assignment:	setting the set of authentication keys for data reading via MIFARE® Classic Protocol (see section 5.7.3)

6.5.24. RFID_Command_GetMIFARE_KeyTable

Input parameter: not used
Output parameter: `TMIFARE_KeyTable *`
Assignment: acquisition of the current set of authentication keys for data reading via MIFARE® Classic Protocol (see section 5.7.3)

6.5.25. RFID_Command_Set_OperationalBaudRate

Input parameter: `long`
Output parameter: not used
Assignment: setting the combination of allowed rates of data exchange between the reader and the RFID-chip (see section 5.4.3)

One of the constants **`eRFID_BaudRate`** is given in the parameter `params`.

6.5.26. RFID_Command_Get_OperationalBaudRate

Input parameter: not used
Output parameter: `long *`
Assignment: acquisition of the current combination of the working rates of data exchange between the reader and the RFID-chip (see section 5.4.3)

6.5.27. RFID_Command_Set_PassivePKD

Input parameter: `char *`
Output parameter: not used
Assignment: setting a full name of the directory, containing a set of PKD files for PA (see section 5.5.2)

Full PKD directory name (in UTF8 format) is given in `params` parameter.

6.5.28. RFID_Command_Get_PassivePKD

Input parameter: not used
Output parameter: `char **`
Assignment: acquisition of full name of the current directory, containing a set of PKD files for PA(see section 5.5.2)

This command initializes the pointer located at the address in `result` parameter, with the pointer to the string in UTF8 format with the current directory full name.

6.5.29. RFID_Command_Set_EAC_PKD

Input parameter:	`char *`
Output parameter:	not used
Assignment:	setting a full name of the directory, containing a set of PKD files for TA (see section 5.5.3)

Full PKD directory name (in UTF8 format) is given in `params` parameter.

6.5.30. RFID_Command_Get_EAC_PKD

Input parameter:	not used
Output parameter:	`char **`
Assignment:	acquisition of full name of the current directory, containing a set of PKD files for TA (see section 5.5.3)

This command initializes the pointer located at the address in `result` parameter, with the pointer to the string in UTF8 format with the current directory full name.

6.5.31. RFID_Command_Get_ReadCardProperties

Input parameter:	not used
Output parameter:	`TRFCardProp *`
Assignment:	acquisition of RFID-chip characteristics, located in the scope of the reader when working in the batch mode (see section 5.7.1)

This command fills `TRFCardProp` structure, located at the address in `result` parameter, by the newly received data from the RFID-chip.

This command cleans the results obtained by previous **`RFID_Command_ReadProtocol3`** and **`RFID_Command_ReadProtocol4`** commands of data reading.

6.5.32. RFID_Command_ReadCardPropertiesExt

Input parameter:	not used
Output parameter:	`TRFID_CardPropertiesExt **`
Assignment:	acquisition of RFID-chip characteristics, located in the scope of the reader when working in the batch mode (see section 5.7.1)

This command returns the pointer to **`TRFID_CardPropertiesExt`** structure, located at the address in `result` parameter, which contains information on RFID-chip's characteristics.

This command cleans the results obtained by previous **RFID_Command_ReadProtocol3** and **RFID_Command_ReadProtocol4** commands of data reading.

6.5.33. RFID_Command_ReadCardPropertiesExt2

Input parameter:	not used
Output parameter:	`TRFChipProperties **`
Assignment:	acquisition of RFID-chip characteristics, located in the scope of the reader (see section 5.7.1)

This command returns the pointer to **TRFChipProperties**, structure, located at the address in `result` parameter, which contains information on RFID-chip's characteristics (for readers with firmware version 21.00 and higher).

6.5.34. RFID_Command_ReadProtocol3

Input parameter:	not used
Output parameter:	`TResultContainerList **`
Assignment:	data reading from the memory of the RFID-chip via ISO/IEC 14443-3 protocol (MIFARE® Classic Protocol) when working in the batch mode (see section 5.7.3)

This command initializes the pointer to **TResultContainerList** structure, located at the address in `result` parameter, with the pointer to the list of container structures with the newly received data from the RFID-chip.

6.5.35. RFID_Command_ReadProtocol4

Input parameter:	`long`
Output parameter:	`TResultContainerList **`
Assignment:	data reading from the memory of the RFID-chip via ISO/IEC 14443-4 protocol when working in the batch mode (see section 5.7.4)

A combination of **eRFID_DataGroups** flags is passed in `params` parameter of this command, specifying the set of read data groups.

This command initializes the pointer to **TResultContainerList** structure, located at the address in `result` parameter, with the pointer to the list of container structures with the newly received data from the RFID-chip.

6.5.36. RFID_Command_CancelReading

Input parameter:	not used
Output parameter:	not used
Assignment:	forced termination of data reading from the memory of the RFID-chip (see sections 5.7.4, 5.8.10)

6.5.37. RFID_Command_DocumentDone

Input parameter:	`long`
Output parameter:	not used
Assignment:	finalization of work with the RFID-chip (see sections 5.4.1, 5.4.6, 5.7.8)

One of **eRFID_ManualChipDetectionMode** values is specified in `params` parameter of this command, defining the further action of RFID-chips search in the scope of the reader antenna.

6.5.38. RFID_Command_IsDocument

Input parameter:	not used
Output parameter:	`bool *`
Assignment:	determination of the current status of the availability of RFID-chip in the scope of the reader (see section 5.4.1)

6.5.39. RFID_Command_ParseRawData

Input parameter:	`TCustomRawDataList *`
Output parameter:	`TResultContainerList **`
Assignment:	analysis of data previously received from the RFID-chip

`params` parameter of this command should contain a pointer to the list of binary representation of the informational data groups contents, reading of which was performed earlier.

This command initializes the pointer to **TResultContainerList** structure, located at the address in `result` parameter, with the pointer to the list of container structures with logically parsed data of information groups.

6.5.40. RFID_Command_ClearResults

Input parameter:	not used
Output parameter:	not used
Assignment:	cleaning memory occupied by the current results of work with electronic document (see sections 5.8.1, 5.9)

6.5.41. RFID_Command_Set_DetectionMode

Input parameter: `long`
Output parameter: not used
Assignment: chip detection mode setting (see section 6.4.45)

`CtrlRF_Auto` or `CtrlRF_Manual` of **`eDataProcessingLevel`** values is given in `params` parameter of this command.

6.5.42. RFID_Command_SetDataProcessingLevel

Input parameter: `long`
Output parameter: not used
Assignment: setting the level of strictness of SDK reaction to detection of discrepancies in the structure of processed data and the errors of execution of different operations (see section 5.2)

One of **`eDataProcessingLevel`** values is given in `params` parameter of this command.

6.5.43. RFID_Command_GetDataProcessingLevel

Input parameter: not used
Output parameter: `long *`
Assignment: acquisition of the current value of the level of strictness of SDK reaction to detection of discrepancies in the structure of the processed data and errors when executing different operations (see section 5.2)

6.5.44. RFID_Command_SetTransferBufferSize

Input parameter: `long`
Output parameter: not used
Assignment: setting the data reading buffer size, activation of the mode of using the extended length reading commands (see section 5.4.4)

6.5.45. RFID_Command_GetTransferBufferSize

Input parameter: not used
Output parameter: `long *`
Assignment: acquisition of the data reading buffer size (see section 5.4.4)

6.5.46. RFID_Command_SetUserDefinedFilesToRead

Input parameter: `TRFID_FilesList *`
Output parameter: not used
Assignment: setting the list of non-standard files to include in the overall reading operation when working in the batch mode (see section 5.7.4)

6.5.47. RFID_Command_Set_DS_Cert_Priority

Input parameter: `long`
Output parameter: not used
Assignment: definition of the priority of using DS-certificates from different sources (see section 5.5.2)

One of **`eDSCertificatePriority`** values is given in `params` parameter of this command.

6.5.48. RFID_Command_Get_DS_Cert_Priority

Input parameter: not used
Output parameter: `long *`
Assignment: acquisition of the current value of the priority of using DS-certificates from different sources (see section 5.5.2)

6.5.49. RFID_Command_Set_TrustedPKD

Input parameter: `long`
Output parameter: not used
Assignment: setting the level of trust to CSCA-certificates from PKD (see section 5.5.2)

A sign of the maximum trust level activation is specified in `params` parameter of this command (`true` or `false`).

6.5.50. RFID_Command_Get_TrustedPKD

Input parameter: not used
Output parameter: `long *`
Assignment: acquisition of the sign of maximum trust level activity to CSCA-certificates from PKD (see section 5.5.2)

6.5.51. RFID_Command_Session_Open

Input parameter: not used
Output parameter: `TRFID_Session **`

Assignment:	opening of the work session with electronic document (see section 5.8.3)

The command initializes the pointer by the address given in `result` parameter with the reference to the created data object of the session work results.

6.5.52. RFID_Command_Session_SelectApplication

Input parameter:	`TRFID_ApplicationID *`
Output parameter:	`TRFID_Session *`
Assignment:	selection of the application within the context of the communication session with electronic document (see section 5.8.9)

6.5.53. RFID_Command_Session_AccessControlProc

Input parameter:	`TRFID_AccessControl_Params *`
Output parameter:	`TRFID_Session *`
Assignment:	authentication or secure data access procedure within the context of the communication session with electronic document (see section 5.8.7)

6.5.54. RFID_Command_Session_ReadFile

Input parameter:	`TRFID_FileID *`
Output parameter:	`TRFID_Session *`
Assignment:	data reading from the file within the context of the communication session with electronic document (see section 5.8.10)

6.5.55. RFID_Command_Session_PA_CheckSO

Input parameter:	`TPA_Params *`
Output parameter:	`TRFID_Session *`
Assignment:	document security object verification within the context of the communication session with electronic document (see section 5.8.12)

6.5.56. RFID_Command_Session_PA_CheckFile

Input parameter:	`TRFID_DataFile *`
Output parameter:	`TRFID_Session *`
Assignment:	file data integrity verification within the context of the communication session with electronic document (see section 5.8.13)

6.5.57. RFID_Command_Session_Close

Input parameter:	`long`
Output parameter:	`TRFID_Session *`
Assignment:	closing of the communication session with electronic document

One of **`eRFID_ManualChipDetectionMode`** values is passed in `params` parameter of this command, determining the further action of search of RFID-chips in the scope of the reader antenna when working in the mode of manual detection of RFID-chip (See description **`RFID_Command_DocumentDone`**).

6.5.58. RFID_Command_Session_ReadMifare

Input parameter:	not used
Output parameter:	`TRFID_Session *`
Assignment:	performance of the data reading procedure via ISO/IEC 14443-3 protocol (MIFARE® Classic Protocol) within the context of the communication session with electronic document (see section 5.8.11)

6.5.59. RFID_Command_Session_SetAccessKey

Input parameter:	`TRFID_AccessKey *`
Output parameter:	`TRFID_Session *`
Assignment:	selection and initialization of the data access key within the context of the communication session with electronic document (see section 5.8.6)

6.5.60. RFID_Command_Session_SetTerminalType

Input parameter:	`TRFID_Terminal *`
Output parameter:	`TRFID_Session *`
Assignment:	setting the configuration of the current terminal within the context of the communication session with electronic document (see section 5.8.4)

6.5.61. RFID_Command_Session_SetProcedureType

Input parameter:	`long`
Output parameter:	`TRFID_Session *`
Assignment:	setting the type of performed authentication procedure within the context of the communication session with electronic document (see section 5.8.5)

One of **eRFID_AuthenticationProcedureType** values is specified in `params` parameter of this command, determining the type of procedure.

6.5.62. RFID_Command_Session_WriteFile

Input parameter:	`TRFID_FileUpdateData *`
Output parameter:	`TRFID_Session *`
Assignment:	the operation of updating the contents of informational data group within the context of the communication session with electronic document (see section 5.8.19)

6.5.63. RFID_Command_Session_Verify

Input parameter:	`long`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of auxiliary data verification within the context of the communication session with electronic document (see section 5.8.18)

One of **eRFID_AuxiliaryDataType** values is specified in `params` parameter of this command, determining the type of verified data.

6.5.64. RFID_Command_Session_Password_ChangePIN

Input parameter:	`char *`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of changing the value of PIN password within the context of the communication session with electronic document (see section 5.8.20)

A pointer to the character string (ASCII) is specified in `params` parameter of this command, determining the new password contents.

6.5.65. RFID_Command_Session_Password_ChangeCAN

Input parameter:	`char *`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of changing the value of CAN password within the context of the communication session with electronic document (see section 5.8.20)

A pointer to the character string (ASCII) is specified in `params` parameter of this command, determining the new password contents.

6.5.66. RFID_Command_Session_Password_UnblockPIN

Input parameter: not used
Output parameter: `TRFID_Session *`
Assignment: the procedure of unblocking PIN password within the context of the communication session with electronic document (see section 5.8.20)

6.5.67. RFID_Command_Session_Password_ActivatePIN

Input parameter: not used
Output parameter: `TRFID_Session *`
Assignment: the procedure of activating PIN password within the context of the communication session with electronic document (see section 5.8.20)

6.5.68. RFID_Command_Session_Password_DeactivatePIN

Input parameter: not used
Output parameter: `TRFID_Session *`
Assignment: the procedure of deactivating PIN password within the context of the communication session with electronic document (see section 5.8.20)

6.5.69. RFID_Command_Session_PA_IsFileCheckAvailable

Input parameter: `TRFID_FileID *`
Output parameter: `TRFID_Session *`
Assignment: check of the presence of specific file hash value in the structure of the detected document security objects within the context of the communication session with electronic document (see sections 5.8.10, 5.8.13)

6.5.70. RFID_Command_Session_eSign_CreatePIN

Input parameter: `TRFID_eSignPINParameters *`
Output parameter: `TRFID_Session *`
Assignment: the procedure of creating *eSign-PIN* password within the context of the communication session with electronic document (see section 5.8.20)

6.5.71. RFID_Command_Session_eSign_ChangePIN

Input parameter: `TRFID_eSignPINParameters *`
Output parameter: `TRFID_Session *`

Assignment:	the procedure of changing *eSign-PIN* password within the context of the communication session with electronic document (see section 5.8.20)

6.5.72. RFID_Command_Session_eSign_UnblockPIN

Input parameter:	`TRFID_eSignPINParameters *`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of unblocking *eSign-PIN* password within the context of the communication session with electronic document (see section 5.8.20)

6.5.73. RFID_Command_Session_eSign_TerminatePIN

Input parameter:	`TRFID_eSignPINParameters *`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of terminating *eSign-PIN* password within the context of the communication session with electronic document (see section 5.8.20)

6.5.74. RFID_Command_Session_eSign_VerifyPIN

Input parameter:	not used
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of verifying *eSign-PIN* password within the context of the communication session with electronic document (see section 5.8.21)

6.5.75. RFID_Command_Session_eSign_GenerateKeyPair

Input parameter:	`TRFID_eSignKeyParameters *`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of creating a pair of cryptographic keys for *`eSign`* application within the context of the communication session with electronic document (see section 5.8.21)

6.5.76. RFID_Command_Session_eSign_TerminateKeyPair

Input parameter:	`TRFID_eSignKeyParameters *`
Output parameter:	`TRFID_Session *`
Assignment:	the procedure of terminating a pair of cryptographic keys for *`eSign`* application within the context of the communication session with electronic document (see section 5.8.21)

6.5.77. RFID_Command_Session_eSign_SignData

Input parameter: `TCustomRawData *`
Output parameter: `TRFID_Session *`
Assignment: the procedure of creating a data digital signature within the context of the communication session with electronic document (see section 5.8.21)

6.5.78. RFID_Command_Session_LoadData

Input parameter: `TCustomRawData *`
Output parameter: `TRFID_Session **`
Assignment: the procedure of creating a session object on the basis of the existing integral block of data (see section 5.8.22)

The command initializes the pointer by the address specified in `result` parameter, with a reference to the created data object with the results of work of virtual session.

6.5.79. RFID_Command_Session_SaveData

Input parameter: `TCustomRawData *`
Output parameter: `TRFID_Session *`
Assignment: the procedure of creating of the integral block of session data (see section 5.8.22)

The command fills the object by the pointer set in `params` parameter, with data of the results of the current session work.

6.5.80. RFID_Command_Get_ProfilerType

Input parameter: `not used`
Output parameter: `long *`
Assignment: request the type of logical data profiler to use with the electronic document in accordance with the requirements of [2] and [3] (default) or [31] (see section 5.2)

6.5.81. RFID_Command_Set_ProfilerType

Input parameter: `long`
Output parameter: `not used`
Assignment: selection of the type of logical data profiler to use with the electronic document in accordance with the requirements of [2] and [3] (default) or [31] (see section 5.2)

In `params` parameter of this command is given one of **`eRFID_SDK_ProfilerType`** values.

6.5.82. RFID_Command_Get_DefaultPACEOption

Input parameter:	`not used`
Output parameter:	`long *`
Assignment:	request the default index of PACE procedure variant (see section 5.8.8)

6.5.83. RFID_Command_Set_DefaultPACEOption

Input parameter:	`long`
Output parameter:	`not used`
Assignment:	definition of the default index of PACE procedure variant (see section 5.8.8)

In `params` parameter of this command is given the index of procedure variant.

6.5.84. RFID_Command_Scenario_Process

Input parameter:	`char *`
Output parameter:	`char **`
Assignment:	conducting the communication session with the electronic document in the scenario operation mode (see section 5.9)

In `params` parameter of this command is given the scenario XML-structure.

XML-representation of **`TRFID_Session`** with the results of the communication session with the electronic document will be referenced by the output parameter.

6.5.85. RFID_Command_Set_OnlineTAToSignDataType

Input parameter:	`long`
Output parameter:	`not used`
Assignment:	definition of the type of data transmitted to the user application on the second step of TA procedure in *Online* and step mode (see section 5.8.15) or in the scenario operation mode (see section 5.9.4.7)

In `params` parameter of this command is given one of **`eRFID_TerminalAuthenticationToSignDataType`** values.

6.5.86. RFID_Command_Get_OnlineTAToSignDataType

Input parameter: `not used`
Output parameter: `long *`
Assignment: request the type of data transmitted to the user application on the second step of TA procedure in *Online* and step mode (see section 5.8.15) or in the scenario operation mode (see section 5.9.4.7)

6.5.87. RFID_Command_Set_Graphics_CompressionRatio

Input parameter: `long`
Output parameter: `not used`
Assignment: setting the compression level of images when working with the respective graphic formats (see section 5.6.3)

6.5.88. RFID_Command_Get_Graphics_CompressionRatio

Input parameter: `not used`
Output parameter: `long`
Assignment: reading the compression level of images when working with the respective graphic formats (see section 5.6.3)

6.5.89. RFID_Command_UseDeviceDriverLog

Input parameter: `bool`
Output parameter: `not used`
Assignment: activation/deactivation of SDK logging on device driver level (see section 5.5.1)

6.5.90. RFID_Command_Session_LoadData_Reparse

Input parameter: `TCustomRawData *`
Output parameter: `TRFID_Session **`
Assignment: the procedure of creating a session object on the basis of the existing integral block of data (see section 5.8.22)

This is an analogue of **`RFID_Command_Session_LoadData`** command (see section 6.5.85) except that there is a repeated logical analisys of the data provided taking place with a composition of a new set of possible notifications.

6.5.91. RFID_Command_Set_UseExternalCSCA

Input parameter:	`long`
Output parameter:	`not used`
Assignment:	limitation of the use of CSCA-certificates submitted by individual data files only (see section 5.5.2)

A sign of the limitation is specified in `params` parameter of this command (`true` or `false`).

6.5.92. RFID_Command_Get_UseExternalCSCA

Input parameter:	`not used`
Output parameter:	`long *`
Assignment:	request the limitation of the use of CSCA-certificates submitted by individual data files only (see section 5.5.2)

6.5.93. RFID_Command_Set_TCC_Params

Input parameter:	`char *`
Output parameter:	`not used`
Assignment:	setting TCC service parameters

The command sets parameters of the TCC service implemented according to the standard BSI TR-03129. A json with the following structure is expected as an input parameter:

```
{
"tccParams":{
    "serviceUrl":"...",
    "pfxCertUrl":"...",
    "pfxPassPhrase":"..."
    }
}
```

serviceUrl — URL of the TCC service;
pfxCertUrl — URL from which a PFX certificate of the service is downloaded;
pfxPassPhrase — PFX certificate passphrase (required if the certificate is protected with a passphrase).