

Epson Windows SDK for Multiple TSE Connections

API Reference for German Fiscal

Precautions

- Unauthorized duplication, copying, reproduction, or modification of any part or all of this document is strictly prohibited.
- Contents of this manual are subject to change without prior notice. Contact us directly for the most recent information.
- Every effort is made to ensure that the contents of this manual are without error. Please contact us if any errors or other issues are found.
- The previous statement notwithstanding, we will not be liable for any negative impact as a result of use.
- Epson shall not be liable for any damages caused as a result of using this product incorrectly, failing to comply with the content of this document, or having repair or modifications performed by third parties other than Epson or those specified by Epson.
- Epson shall not be liable for any issues as a result of installing optional parts or consumables that are not genuine Epson parts or parts certified by Epson.

Trademarks

EPSON, EXCEED YOUR VISION, and ESC/POS are registered trademarks of Seiko Epson Corporation.

ESC/POS® Command System

Epson has embarked on a global initiative by developing ESC/POS, a unique POS printer command system. ESC/ POS contains a wealth of unique commands, many of which are patent-protected. Our system enables the configuration of versatile POS systems with a high level of scalability. In addition to being compatible with most Epson POS printers and displays, the flexibility provided by this unique control system facilitates ease of future upgrades. This functionality and convenience of use are appreciated around the world.

© Seiko Epson Corporation 2020. All rights reserved.

About this Document

Purpose of this Document

This document provides the information to describe the functionality provided by the Windows SDK DLL.

Contents

About this Document	3
Contents	4

About the Windows SDK DLL	6
Description	6
Support	6
Usage Limitations	6

API Reference	7
---------------------	---

TSE API List	7
GetTseDeviceList	8
TSEConnect	9
TSEDisconnect	11
TSEOpenDevice	12
TSECloseDevice	13
Setup	14
RunTSESelfTest	16
FactoryReset	17
GetRawStorageInfo	18
GetStorageInfo	19
GetRawStorageSmartInfo	20
GetStorageSmartInfo	21
GetStartedTransactionList	22
GetLastTransactionResponse	23
GetMaxNumberOfClients	25
GetCurrentNumberOfClients	26
GetMaxNumberOfTransactions	27
GetCurrentNumberOfTransactions	28
ExportCertificate	29
ExportSerialNumber	30
AuthenticateUser	31
LogOut	33
ChangePin	34
UpdateTime	36
DisableSecureElement	37
RegisterTSEToHost	38
RegisterSecretKey	39
RegisterClient	40
DeregisterClient	41
LockTse	42
UnlockTse	43

EnableExportIfCspTestFails	44
DisableExportIfCspTestFails.....	45
SetTimeoutInterval.....	46
GetTimeoutInterval	47
GetAuthenticatedUserList.....	48
GetRegisteredClientList.....	50
StartTransaction.....	51
UpdateTransaction.....	54
FinishTransaction.....	56
ExportDataFilteredByTransactionNumberAndClientId	59
ExportDataFilteredByTransactionNumber	61
ExportDataFilteredByTransactionNumberInterval	62
ExportDataFilteredByTransactionNumberInterval.....	64
ExportDataFilteredByPeriodOfTimeAndClientId	66
ExportDataFilteredByPeriodOfTime	68
ExportData.....	70
FinalizeExport.....	71
CancelExport	72
ChangePuk.....	73
UnblockUser.....	75
TSEMemDealloc.....	77
TSEMemDeallocTseInfoObject.....	78
TSESetLog	79
TSEGetErrorDescription.....	81
EPSON_TSE_INFO	82
EPSON_TSE_SMART_INFO	86
Error code.....	89

About the Windows SDK DLL

Description

The Windows SDK for Multiple TSE Connections DLL (EpsonTseLibCMulti.dll) is a C-based module that implements the functionalities to operate and manage multiple Epson TSEs.

To manage connections to multiple TSEs, this DLL maintains a list of *tseHandles*, where each *tseHandle* is used to identify a specific TSE connection so that further operation can be performed on the TSE. A *tseHandle* is created when a connection to a TSE is successfully established (using `TSEConnect()`) and destroyed when the connection to the TSE is successfully terminated (using `TSEDisconnect()`).

Support

The following is the list of devices supported by the Windows SDK DLL with the corresponding protocol and communication interface to which the devices can work with.

Device	Protocol and Interface
Epson printers that support German Fiscal (TSE).	ESC/POS protocol via Serial connection ESC/POS protocol via Socket (TCP) connection ePOS-Device XML via Socket (TCP) connection
EPS TSE Server 3/8	ePOS-Device XML via Socket (TCP) connection
Epson Windows TSE driver	ePOS-Device XML via Socket (TCP) connection

Usage Limitations

The use of the DLL is intended mainly for applications or POS systems that are required to operate the Epson TSEs.

The DLL and its corresponding source files are not guaranteed free from error, defects, and bugs. Any modifications in the DLL binary file or source files made by the customer shall be the responsibility of the customer.

The DLL is not guaranteed to be thread-safe.

The DLL is not intended for use in applications that require extremely high levels of reliability and safety such as in aerospace equipment, trunk-line communications equipment, nuclear power control equipment, and medical equipment. Consider your usage environment and requirements carefully before using this product in such applications.

API Reference

TSE API List

This chapter describes the APIs provided by Epson Windows SDK for German Fiscal (TSE).

GetTseDeviceList

Gets the list of TSE device IDs connected to an EPS TSE Server.

Syntax

```
short int GetTseDeviceList(
    unsigned char      *ipAddress,
    unsigned long int  ipAddressLength,
    unsigned char      **tseDeviceList,
    unsigned long int  *tseDeviceListSize);
```

Parameter

[IN] *ipAddress*

Pointer to a buffer that contains the IP Address of the target TSE Server.
Cannot be a null pointer.

[IN] *ipAddressLength*

The size of the buffer containing the IP Address of the TSE Server.

[OUT] *tseDeviceList*

Pointer to the buffer that receives the list of TSE device IDs connected to the specified TSE server. The TSE Devices IDs are separated by a null character.

The memory block needed to receive the TSE Device List is allocated internally by this function and can be deallocated using the TSEMemDealloc.

Cannot be a null pointer.

[OUT] *tseDeviceListSize*

Pointer that receives the size of the memory block allocated for *tseDeviceList*.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

TSEConnect

Establishes and initializes a connection to a target TSE.

- ❑ When connection is successfully created, this function returns a descriptor (of type `void*`) that references the connection to the target TSE. The descriptor can be used to identify the same TSE connection on future operations.
- ❑ If a connection to a TSE has already been established and still active, then establishing a new connection to the same TSE will not be successful. The existing connection has to be disconnected first (using `TSEDisconnect()`) before making a new connection to the same TSE.

Syntax

```
short int TSEConnect(
    unsigned short int method,
    unsigned char      *portId,
    unsigned long int   portIdLength,
    unsigned char      *tseId,
    unsigned long int   tseIdLength,
    void                **tseHandlePtr);
```

Parameter

[IN] *method*

The method to communicate with the host device.

Value	Description
ACCESS_BY_SERIAL_ESCPOS	Using ESC/POS protocol via Serial connection
ACCESS_BY_NETWORK_ESCPOS	Using ESC/POS protocol via Socket (TCP) connection
ACCESS_BY_EPOS_DEVICE	Using ePOS-Device XML via Socket (TCP) connection

[IN] *portId*

Pointer to the buffer that contains the string that identifies the device that hosts the TSE. If the method is set to serial connection, then this corresponds to the COM port number (e.g. COM1) of the device. If the method is set to Socket (TCP) connection, then this corresponds to the IP Address.

Cannot be a null pointer.

[IN] *portIdLength*

The size of the buffer that contains the portId.

[IN] *tseld*

Pointer to the buffer that contains the target TSE Device ID.

If the buffer contains an empty string, then the default TSE Device ID ("local_TSE") will be set.

Cannot be a null pointer.

[IN] *tseldLength*

The size of the buffer that contains the target TSE Device ID.

[OUT] *tseHandlePtr*

Pointer that receives the descriptor (of type `void*`) that references the established TSE connection. Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

TSEDisconnect

Terminates the connection to the target TSE.

- ❑ The function destroys the `tseHandle` when the TSE connection is successfully terminated.

Syntax

```
short int TSEDisconnect(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

TSEOpenDevice

Enables sending commands to the TSE.

Syntax

```
short int TSEOpenDevice(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

TSECloseDevice

Disables sending commands to the TSE.

Syntax

```
short int TSECloseDevice(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

Setup

Sets up and initializes the TSE to make it ready for use.

Sets PUK, Admin PIN, TimeAdmin PIN in the TSE.

- ❑ Internally executes TSE unlock (see `UnlockTse`), enables export after a CSP test failure in TSE (see `EnableExportIfCspTestFails`), and runs TSE self-test (see `RunTSESelfTest`).
- ❑ Internally registers the printer serial number as a client ID in the TSE.

Syntax

```
short int Setup(
    void                *tseHandle,
    unsigned char       *adminPin,
    unsigned long int   adminPinLength,
    unsigned char       *clientIdPin,
    unsigned long int   clientIdPinLength,
    unsigned char       *puk,
    unsigned long int   pukLength);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **adminPin**

Pointer to the buffer that contains the new PIN to be set for Admin privileges.
Cannot be a null pointer.

[IN] **adminPinLength**

The size of the buffer that contains the new PIN to be set for Admin privileges.

[IN] **clientIdPin**

Pointer to the buffer that contains the new PIN to be set for TimeAdmin privileges.
Cannot be a null pointer.

[IN] **clientIdPinLength**

The size of the buffer that contains the new PIN to be set for TimeAdmin privileges.

[IN] **puk**

Pointer to the buffer that contains the new PUK to be set.
Cannot be a null pointer.

[IN] **pukLength**

The size of the buffer that contains the new PUK to be set.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

RunTSESelfTest

Runs the self-test for TSE.

- ❑ The self-test checks to see if the internal module of the TSE works properly.
- ❑ The printer automatically runs the self-test of the TSE after each power-on, so basically, the user does not need to run the self-test manually.

However, if the printer is used for 25 hours or longer without being turned off, the user needs to run the self-test using this function before the time for continuous run of the printer reaches 25 hours.

- ❑ To acquire the remaining time before the self-test is required, use "timeUntilNextSelfTest" of GetStorageInfo.

Syntax

```
short int RunTSESelfTest(
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

FactoryReset

Resets the TSE to factory settings.

- ❑ This function also internally runs TSE self-test (see RunTSESelfTest).
- ❑ **Please note that this function exists only for development purposes and will not be available for the final TSEs.**

Syntax

```
short int FactoryReset(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetRawStorageInfo

Acquires the German fiscal element (TSE) information in raw JSON string.

Syntax

```
short int GetRawStorageInfo(
    void                *tseHandle,
    unsigned char       **rawTseInfo,
    unsigned long int    *rawTseInfoLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *rawTseInfo*

Pointer to the buffer that receives the TSE information in raw JSON string.
The memory block needed to receive the TSE information is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.
Cannot be a null pointer.

[OUT] *rawTseInfoLength*

Pointer that receives the size of the memory block allocated for `rawTseInfo`.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

GetStorageInfo

Acquires the German fiscal element (TSE) information.

Syntax

```
short int GetStorageInfo(
    void                *tseHandle,
    EPSON_TSE_INFO      *tseInfo);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *tseInfo*

Pointer to the struct object that contains the TSE Information (See `EPSON_TSE_INFO` for more details). The memory blocks needed to receive the `serialNumber`, `tsePublicKey`, and `rawTseInfo` of the `EPSON_TSE_INFO` are allocated internally by this function and can be deallocated using the `TSEMemDeallocTseInfoObject`. Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully. Otherwise, the function returns an Error code.

GetRawStorageSmartInfo

Acquires the German fiscal element (TSE) storage smart information in raw JSON string.

Syntax

```
short int GetRawStorageSmartInfo(
    void                *tseHandle,
    unsigned char       **rawSmartInfo,
    unsigned long int    *rawSmartInfoLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *rawSmartInfo*

Pointer to the buffer that receives the TSE smart information in raw JSON string.
The memory block needed to receive the TSE information is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.
Cannot be a null pointer.

[OUT] *rawSmartInfoLength*

Pointer that receives the size of the memory block allocated for `rawSmartInfo`.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

GetStorageSmartInfo

Acquires the German fiscal element (TSE) storage smart information.

Syntax

```
short int GetStorageSmartInfo(
    void                *tseHandle,
    EPSON_TSE_SMART_INFO *smartInfo);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *smartInfo*

Pointer to the struct object that contains the TSE smart information (See `EPSON_TSE_SMART_INFO` for more details).

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetStartedTransactionList

Acquires a list of started (and incomplete) transactions.

Syntax

```
short int GetStartedTransactionList(
    void                *tseHandle,
    unsigned char       *clientId,
    unsigned long int   clientIdLength,
    unsigned long int   **startedTransactionList,
    unsigned long int   *startedTransactionListSize);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **clientId**

Pointer to the buffer that contains the client ID of the started transaction/s.
When an empty string "" is specified, a list of transaction numbers of all started and incomplete transactions is returned regardless of the client ID.
Cannot be a null pointer.

[IN] **clientIdLength**

The size of the buffer that contains the client ID of the started transaction/s.

[OUT] **startedTransactionList**

Pointer to the buffer that receives the list of transaction numbers of started (but incomplete) transaction/s.
The memory block needed to receive the transaction numbers is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.
Cannot be a null pointer.

[OUT] **startedTransactionListSize**

Pointer that receives the number of started (but incomplete) transactions.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetLastTransactionResponse

Acquires the result of the last transaction written in the TSE for the specified client ID.

Syntax

```
short int GetLastTransactionResponse(
    void                *tseHandle,
    unsigned char        *clientId,
    unsigned long int    clientIdLength,
    unsigned long int    *transactionNumber,
    unsigned long int    *logTime,
    unsigned long int    *signatureCounter,
    unsigned char        **serialNumber,
    unsigned long int    *serialNumberLength,
    unsigned char        **signatureValue,
    unsigned long int    *signatureValueLength);
);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *clientId*

Pointer to a buffer that contains the client ID of the latest transaction result to be queried. Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID of the latest transaction result to be queried.

[OUT] *transactionNumber*

Pointer that receives the transaction counter of the latest transaction corresponding to the specified client ID. Cannot be a null pointer.

[OUT] *logTime*

Pointer that receives the value that corresponds to the date and time of the latest transaction log corresponding to the specified client ID. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC. Cannot be a null pointer.

[OUT] *signatureCounter*

Pointer that receives the signature counter of the latest transaction log corresponding to the specified client ID. Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetMaxNumberOfClients

Acquires the maximum number of clients that can be registered.

Syntax

```
short int GetMaxNumberOfClients(  
    void *tseHandle,  
    unsigned long int *maxNumberClients);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *maxNumberClients*

Pointer that receives the maximum number of clients that can be registered.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetCurrentNumberOfClients

Acquires the number of clients that is currently registered.

Syntax

```
short int GetCurrentNumberOfClients(  
    void *tseHandle,  
    unsigned long int *currentNumberClients);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *currentNumberClients*

Pointer that receives the number of clients that is currently registered.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetMaxNumberOfTransactions

Acquires the maximum number of started transactions.

- This corresponds to the number of transactions that can be started in parallel.

Syntax

```
short int GetMaxNumberOfTransactions(
    void *tseHandle,
    unsigned long int *maxNumberTransactions);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *maxNumberTransactions*

Pointer that receives the maximum number of started transactions.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

GetCurrentNumberOfTransactions

Acquires the number of transactions that have not been finished, yet.

- ❑ If this equals Max Started Transactions, no new transactions can be started until at least one transaction has been finished.
- ❑ Returns the correct value only when TSE is unlocked.

Syntax

```
short int GetCurrentNumberOfTransactions (
    void                *tseHandle,
    unsigned long int    *currentNumberTransactions);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *currentNumberTransactions*

Pointer that receives the number of transactions that have not been finished.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

ExportCertificate

Acquires the certificate that can be used to verify the signatures of all Log Messages created by the TSE.

- ❑ The returned data is a single PEM file, which contains multiple certificates, since the TSE's certificate is signed by other certificates.
- ❑ To verify the signature, only the leaf certificate (the first one in the PEM file) is required.
- ❑ To make sure the leaf certificate is genuine, the next certificate in the file can be used to verify the previous certificate until the last certificate is about to be checked, which will be the root certificate that must be trusted by the system.

Syntax

```
short int ExportCertificate(
    void                *tseHandle,
    unsigned char       **certificate,
    unsigned long int    *certificateLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *certificate*

Pointer to a buffer that receives the certificate that can be used to verify the signatures of all Log Messages created by the Germany fiscal element.

The memory block needed to receive the certificate is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.

Cannot be a null pointer.

[OUT] *certificateLength*

Pointer that receives the size of the memory block allocated for `certificate`.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportSerialNumber

Acquires the serial number of the TSE.

- ❑ Raw SHA-256 hash over the public key that belongs to the private key generating signatures. This can be used as Germany fiscal element (TSE) unique ID.

Syntax

```
short int ExportSerialNumber(
    void                *tseHandle,
    unsigned char       **serialNumber,
    unsigned long int    *serialNumberLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *serialNumber*

Pointer to a buffer that receives the raw binary form of the serial number of the TSE.
The memory block needed to receive the serial number is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.
Cannot be a null pointer.

[OUT] *serialNumberLength*

Pointer that receives the size of the memory block allocated for `serialNumber`.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

AuthenticateUser

Logs in to the TSE with Admin or TimeAdmin privileges.

- ❑ Enables the use of functions with Admin or TimeAdmin privileges.
- ❑ Automatically log out when left (functions that require privilege are not used) for a certain time (default 15 minutes for Admin and 8 hours for TimeAdmin). This timeout period can be set using `SetTimeoutInterval`.

Syntax

```
short int AuthenticateUser(
    void                *tseHandle,
    unsigned char       *userId,
    unsigned long int   userIdLength,
    unsigned char       *pin,
    unsigned long int   pinLength,
    unsigned char       *secretKey,
    unsigned long int   secretKeyLength,
    short int           *remainingRetries);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *userId*

Pointer to the buffer that contains the user ID to log-in as Admin or TimeAdmin.

If the target is to log in as Admin, then the user ID must be "Administrator", otherwise, the user ID has to be a valid client ID to log in as TimeAdmin.

Cannot be a null pointer.

[IN] *userIdLength*

The size of the buffer containing the user ID.

[IN] *pin*

Pointer to the buffer that contains the PIN to log-in as Admin or TimeAdmin.

Cannot be a null pointer.

[IN] *pinLength*

The size of the buffer containing the PIN.

[IN] *secretKey*

Pointer to the buffer that contains the secret key used to calculate the hash value for user authentication.

Cannot be a null pointer.

[IN] *secretKeyLength*

The size of the buffer containing the secret key.

[OUT] *remainingRetries*

Pointer to receive the number of input PIN retries remaining.

PIN/PUK authentication can fail up to 3 times. This will be -1 when PIN authentication is successful or when an error other than authentication failure is encountered.

Reduces by 1 on each PIN authentication failure.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

LogOut

Logs out a user logged in with Admin or TimeAdmin privileges.

- ❑ To use this function, the user must be logged in with the corresponding Admin or TimeAdmin privileges.

Syntax

```
short int LogOut(
    void                *tseHandle,
    unsigned char       *userId,
    unsigned long int   userIdLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *userId*

Pointer to the buffer that contains the user ID of the user logged-in as Admin or TimeAdmin. If the target is to log out from Admin privileges, then the user ID must be "Administrator", otherwise, the user ID has to be a valid client ID to log out as TimeAdmin. Cannot be a null pointer.

[IN] *userIdLength*

The size of the buffer containing the user ID.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ChangePin

Changes the Admin or TimeAdmin PIN.

- ❑ To use this function, the user must be logged in with the corresponding Admin or TimeAdmin privileges.

Syntax

```
short int ChangePin(
    void                *tseHandle,
    unsigned char        *userId,
    unsigned long int    userIdLength,
    unsigned char        *oldPin,
    unsigned long int    oldPinLength,
    unsigned char        *newPin,
    unsigned long int    newPinLength,
    short int            *remainingRetries);
```

Parameter

[IN] **tseHandle**

The descriptor (of type void*) that identifies the target TSE connection. The descriptor is created by TSEConnect().

[IN] **userId**

Pointer to the buffer that contains the user ID logged-in with Admin or TimeAdmin privileges. If the target is to change the Admin PIN, then the user ID must be "Administrator", otherwise, the user ID has to be a valid client ID to change the TimeAdmin PIN. Cannot be a null pointer.

[IN] **userIdLength**

The size of the buffer containing the user ID.

[IN] **oldPin**

Pointer to the buffer that contains the current Admin or TimeAdmin PIN.

[IN] **oldPinLength**

The size of the buffer containing the current PIN.

[IN] **newPin**

Pointer to the buffer that contains the new Admin or TimeAdmin PIN to be set.

[IN] **newPinLength**

The size of the buffer containing the new PIN.

[OUT] *remainingRetries*

Pointer to receive the number of input PIN retries remaining.

PIN authentication can fail up to 3 times. This will be -1 when PIN authentication is successful or when an error other than authentication failure occurred.

Reduces by 1 on each PIN authentication failure.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

UpdateTime

Updates the TSE time.

- ❑ Updates the TSE time using the UTC time specified from POS.
- ❑ This function must be executed before starting to write transactions.
- ❑ To use this function, the user must be logged in with Admin or TimeAdmin privileges.

Syntax

```
short int UpdateTime(
    void                *tseHandle,
    unsigned char       *userId,
    unsigned long int   userIdLength,
    unsigned long int   newDateTime,
    bool                updateForFirstTime);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **userId**

Pointer to the buffer that contains the user ID logged-in with Admin or TimeAdmin privileges. Cannot be a null pointer.

[IN] **userIdLength**

The size of the buffer containing the user ID.

[IN] **newDateTime**

The value that corresponds to the date and time to be set to the TSE to synchronize with the POS system's date and time. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.

[IN] **updateForFirstTime**

If true, updates the TSE time only when `UpdateTime` is called the first time and skips updating the TSE time when `UpdateTime` is called the second and subsequent times.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

DisableSecureElement

Disables TSE use.

- ❑ New transactions cannot be written after executing this function.
- ❑ Can be executed only when no incomplete transactions exist.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int DisableSecureElement(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

RegisterTSEToHost

Starts-up the TSE after replacing printer.

- ❑ Must be executed whenever the combination of the TSE and printer is changed.
- ❑ Internally registers the printer serial number as a client ID in the TSE.
- ❑ Internally runs the TSE self-test.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int RegisterTSEToHost(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

RegisterSecretKey

Registers the secret key.

- ❑ Registers the key used to calculate the hash value required for user/host authentication when logging in.
- ❑ This setting is persisted across power cycles.
- ❑ Default value is "EPSONKEY", if RegisterSecretKey is never executed.
- ❑ The secret key must be registered after installation or when replacing the printer.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int RegisterSecretKey(
    void                *tseHandle,
    unsigned char        *secretKey,
    unsigned long int    secretKeyLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *secretKey*

Pointer to the buffer that contains the secret key to be set to calculate the hash value for user/host authentication.
Cannot be a null pointer.

[IN] *secretKeyLength*

The size of the buffer containing the secret key.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

RegisterClient

Registers a client ID to be used in the TSE.

- ❑ A total of 100 client IDs can be registered in the TSE. However, as one ID is used internally by the printer or the host device, 99 client IDs can be registered by the user.
- ❑ Only client IDs registered in the TSE can be used for transactions.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int RegisterClient(
    void                *tseHandle,
    unsigned char       *newClientId,
    unsigned long int   newClientIdLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *newClientId*

Pointer to the buffer that contains the new client ID to be registered.
 The string must be represented as an ASN.1 PrintableString, which restricts the allowed characters to the following: A-Za-z0-9'()+,-./:=? and the space character.
 As strings that include "EPSON" are reserved by the printer, registration by the user is prohibited.
 Cannot be a null pointer.

[IN] *newClientIdLength*

The size of the buffer containing the new client ID.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
 Otherwise, the function returns an Error code.

DeregisterClient

Deregisters a client ID registered in the TSE.

- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int DeregisterClient(
    void *tseHandle,
    unsigned char *clientId,
    unsigned long int clientIdLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *clientId*

Pointer to the buffer that contains the new client ID to be deregistered.
As strings that include "EPSON" are reserved by the printer, deleting by the user is prohibited.
Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID.

Return value

The function return `EXECUTION_OK (0)` if executed successfully.

Otherwise, the function returns an Error code.

LockTse

Locks the TSE.

- ❑ Transactions and export functions are not available when the TSE is locked. In fact, only functions belonging to User Authentication can be used.
- ❑ This setting is persisted across power cycles.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int LockTse(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

UnlockTse

Unlocks the TSE.

- ❑ Transactions and export functions are not available when the TSE is locked. In fact, only functions belonging to User Authentication can be used.
- ❑ This setting is persisted across power cycles.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int UnlockTse(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

EnableExportIfCspTestFails

Enables log data in the TSE to be exported even if the CSP (Crypto Service Provider) test fails.

* The CSP test is executed in TSE self-test.

- ❑ If the CSP test fails due to a security module failure, log data cannot be exported by default.
- ❑ Only ExportData can be executed with this command. (Filter-related exports cannot be executed.)
- ❑ This setting is valid only when TSE self-test including the CSP test is successful. That is, this setting is invalid after the CSP test fails due to a security module failure.
- ❑ This setting is persisted across power cycles.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int EnableExportIfCspTestFails (
    void                *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

DisableExportIfCspTestFails

Disables log data in the TSE to be exported if the CSP (Crypto Service Provider) test fails.

- ❑ This setting is valid only when TSE self-test including the CSP test is successful. That is, this setting is invalid after the CSP test fails due to a security module failure.
- ❑ This setting is persisted across power cycles.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int DisableExportIfCspTestFails(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

SetTimeoutInterval

Sets the timeout period.

- ❑ This setting is persisted across power cycles.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int SetTimeoutInterval(
    void *tseHandle,
    unsigned long int timeoutIntervalForAdmin,
    unsigned long int timeoutIntervalForTimeAdmin,
    unsigned long int timeoutIntervalForExport);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *timeoutIntervalForAdmin*

Timeout period for Admin auto logout.

Valid range: 900 to 7200 [sec] (15 minutes to 2 hours)

If 0 is specified, then the default value will be used which is 900 [sec].

[IN] *timeoutIntervalForTimeAdmin*

Timeout period for TimeAdmin auto logout.

Valid range: 900 to 86400 [sec] (15 minutes to 24 hours).

If 0 is specified, then the default value will be used which is 28800 [sec].

[IN] *timeoutIntervalForExport*

Export timeout period.

Valid range: 100 to 330 [sec].

If 0 is specified, then the default value will be used which is 100 [sec].

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

GetTimeoutInterval

Acquires the timeout period.

- ❑ This setting is persisted across power cycles.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int GetTimeoutInterval(
    void                *tseHandle,
    unsigned long int    *timeoutIntervalForAdmin,
    unsigned long int    *timeoutIntervalForTimeAdmin,
    unsigned long int    *timeoutIntervalForExport);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *timeoutIntervalForAdmin*

Pointer to the buffer that receives the timeout period for Admin auto logout.
Valid range: 900 to 7200 [sec] (15 minutes to 2 hours). Default is 900 [sec].
Cannot be a null pointer.

[OUT] *timeoutIntervalForTimeAdmin*

Pointer to the buffer that receives the timeout period for TimeAdmin auto logout.
Valid range: 900 to 86400 [sec] (15 minutes to 24 hours). Default is 28800 [sec].
Cannot be a null pointer.

[OUT] *timeoutIntervalForExport*

Pointer to the buffer that receives the Export timeout period.
Valid range: 100 to 330 [sec]. Default is 100 [sec].
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

GetAuthenticatedUserList

Acquires a list of logged-in (authenticated) user IDs and client IDs.

- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int GetAuthenticatedUserList(
    void                *tseHandle,
    unsigned char        *userRole,
    unsigned long int    userRoleLength,
    unsigned char        **AuthenticatedUserList,
    unsigned long int    *AuthenticatedUserListLength);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **userRole**

Pointer to the buffer that contains the user type.
Only "Admin" or "TimeAdmin" are valid.
Cannot be a null pointer.

[IN] **userRoleLength**

The size of the buffer containing the user type.

[OUT] **AuthenticatedUserList**

Pointer to the buffer that receives the list of logged-in user IDs.
The user IDs in the list are delimited by a null character.
The memory block needed to receive the user ID list is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.
Cannot be a null pointer.

[OUT] **AuthenticatedUserListLength**

Pointer that receives the size of the memory block allocated for `AuthenticatedUserList`.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

GetRegisteredClientList

Acquires a list of client IDs registered in the TSE.

- ❑ Although the client ID "EPSONXXXXXXXX" is included, the XXXXXXXX part used inside the printer is the printer serial number.
- ❑ To use this function, the user must be logged in with Admin privileges.

Syntax

```
short int GetRegisteredClientList(
    void                *tseHandle,
    unsigned char       **registeredClientList,
    unsigned long int    *registeredClientListLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type void*) that identifies the target TSE connection. The descriptor is created by TSEConnect().

[OUT] *registeredClientList*

Pointer to the buffer that receives the list of client IDs registered in the TSE.

The client IDs in the list are delimited by a null character.

The memory block needed to receive the client ID list is allocated internally by this function and can be deallocated using the TSEMemDealloc.

Cannot be a null pointer.

[OUT] *registeredClientListLength*

Pointer that receives the size of the memory block allocated for registeredClientList.

Cannot be a null pointer.

Return value

The function return EXECUTION_OK (0) if executed successfully.

Otherwise, the function returns an Error code.

StartTransaction

Starts a transaction.

- ❑ It is recommended to verify if the transaction is successful in the TSE side if a connection error occurs while executing this function (see `GetLastTransactionResponse` to check the last transaction written in the TSE).
- ❑ To use this function, the client must be logged in with TimeAdmin privileges.

Syntax

```
short int StartTransaction(
    void                *tseHandle,
    unsigned char       *clientId,
    unsigned long int   clientIdLength,
    unsigned char       *processData,
    unsigned long int   processDataLength,
    unsigned char       *processType,
    unsigned long int   processTypeLength,
    unsigned char       *additionalData,
    unsigned long int   additionalDataLength,
    unsigned long int   *transactionNumber,
    unsigned long int   *logTime,
    unsigned char       **serialNumber,
    unsigned long int   *serialNumberLength,
    unsigned long int   *signatureCounter,
    unsigned char       **signatureValue,
    unsigned long int   *signatureValueLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *clientId*

Pointer to the buffer that contains the client ID logged in with TimeAdmin privileges.
Only client IDs registered in TSE are valid.
Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID.

[IN] *processData*

Pointer to the buffer that contains the raw Transaction data.
 The content of the transaction data is decided by the calling application.
 Cannot be a null pointer.

[IN] *processDataLength*

The size of the buffer containing the raw transaction data.

[IN] *processType*

Pointer to the buffer that contains the type of transaction.
 The string must be representable as an ASN.1 PrintableString, which restricts the allowed characters to the following: A-Za-z0-9'()+, -./:=? and the space character.
 The content of the transaction data is decided by the calling application.
 Cannot be a null pointer.

[IN] *processTypeLength*

The size of the buffer containing the type of transaction.

[IN] *additionalData*

Pointer to the buffer that contains the additional data.
 The content of the transaction data is decided by the calling application.
 Invalid for German fiscal element (TSE) model "TSE1", in which case the value is ignored, and an empty string is recommended.
 Cannot be a null pointer.

[IN] *additionalDataLength*

The size of the buffer containing the additional data.
 Invalid for German fiscal element (TSE) model "TSE1", in which case a value of 0 for this parameter is recommended.

[OUT] *transactionNumber*

Pointer to the buffer that receives the transaction number.
 Cannot be a null pointer.

[OUT] *logTime*

Pointer to the buffer that receives the value that corresponds to the date and time the log was created. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.
 Cannot be a null pointer.

[OUT] *serialNumber*

Pointer to the buffer that receives the raw binary form of the serial number of the TSE.
 The memory block needed to receive the serial number is allocated internally by this function and can be deallocated using the TSEMemDealloc.
 Cannot be a null pointer.

[OUT] *serialNumberLength*

Pointer that receives the size of the memory block allocated for `serialNumber`.
 Cannot be a null pointer.

[OUT] *signatureCounter*

Pointer to the buffer that receives the current signature counter value.

Cannot be a null pointer.

[OUT] *signatureValue*

Pointer to the buffer that receives the raw binary form of the signature value.

The memory block needed to receive the serial number is allocated internally by this function and can be deallocated using the TSEMemDealloc.

Cannot be a null pointer.

[OUT] *signatureValueLength*

Pointer that receives the size of the memory block allocated for *signatureValue*.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

UpdateTransaction

Updates a transaction.

- ❑ It is recommended to verify if the transaction is successful in the TSE side if a connection error occurs while executing this function (see `GetLastTransactionResponse` to check the last transaction written in the TSE).
- ❑ To use this function, the client must be logged in with TimeAdmin privileges.

Syntax

```
short int UpdateTransaction(
    void                *tseHandle,
    unsigned char       *clientId,
    unsigned long int   clientIdLength,
    unsigned long int   transactionNumber,
    unsigned char       *processData,
    unsigned long int   processDataLength,
    unsigned char       *processType,
    unsigned long int   processTypeLength,
    unsigned long int   *logTime,
    unsigned char       **signatureValue,
    unsigned long int   *signatureValueLength,
    unsigned long int   *signatureCounter);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *clientId*

Pointer to the buffer that contains the client ID logged in with TimeAdmin privileges.
Only client IDs registered in TSE are valid.
Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID.

[IN] *transactionNumber*

Transaction number of the started transaction to be updated.

[IN] *processData*

Pointer to the buffer that contains the raw Transaction data.
The content of the transaction data is decided by the calling application.
Cannot be a null pointer.

[IN] *processDataLength*

The size of the buffer containing the raw transaction data.

[IN] *processType*

Pointer to the buffer that contains the type of transaction.

The string must be representable as an ASN.1 PrintableString, which restricts the allowed characters to the following: A-Za-z0-9'()+, -./:=? and the space character.

The content of the transaction data is decided by the calling application.

Cannot be a null pointer.

[IN] *processTypeLength*

The size of the buffer containing the type of transaction.

[OUT] *logTime*

Pointer to the buffer that receives the value that corresponds to the date and time the log was created. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.

Cannot be a null pointer.

[OUT] *signatureValue*

Pointer to the buffer that receives the raw binary form of the serial number of the TSE.

The memory block needed to receive the serial number is allocated internally by this function and can be deallocated using the TSEMemDealloc.

Cannot be a null pointer.

[OUT] *signatureValueLength*

Pointer that receives the size of the memory block allocated for *signatureValue*.

Cannot be a null pointer.

[OUT] *signatureCounter*

Pointer to the buffer that receives the current signature counter value.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

FinishTransaction

Ends a transaction.

- ❑ It is recommended to verify if the transaction is successful in the TSE side if a connection error occurs while executing this function (see `GetLastTransactionResponse` to check the last transaction written in the TSE).
- ❑ To use this function, the client must be logged in with TimeAdmin privileges.

Syntax

```
short int FinishTransaction(
    void                *tseHandle,
    unsigned char        *clientId,
    unsigned long int    clientIdLength,
    unsigned long int    transactionNumber,
    unsigned char        *processData,
    unsigned long int    processDataLength,
    unsigned char        *processType,
    unsigned long int    processTypeLength,
    unsigned char        *additionalData,
    unsigned long int    additionalDataLength,
    unsigned long int    *logTime,
    unsigned char        **signatureValue,
    unsigned long int    *signatureValueLength,
    unsigned long int    *signatureCounter);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *clientId*

Pointer to the buffer that contains the client ID logged in with TimeAdmin privileges.
Only client IDs registered in TSE are valid.
Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID.

[IN] *transactionNumber*

Transaction number of the started transaction to be ended.

[IN] *processData*

Pointer to the buffer that contains the raw Transaction data.
 The content of the transaction data is decided by the calling application.
 Cannot be a null pointer.

[IN] *processDataLength*

The size of the buffer containing the raw transaction data.

[IN] *processType*

Pointer to the buffer that contains the type of transaction.
 The string must be representable as an ASN.1 PrintableString, which restricts the allowed characters to the following: A-Za-z0-9'()+, -./:=? and the space character.
 The content of the transaction data is decided by the calling application.
 Cannot be a null pointer.

[IN] *processTypeLength*

The size of the buffer containing the type of transaction.

[IN] *additionalData*

Pointer to the buffer that contains the additional data.
 The content of the transaction data is decided by the calling application.
 Invalid for German fiscal element (TSE) model "TSE1", in which case the value is ignored, and an empty string is recommended.
 Cannot be a null pointer.

[IN] *additionalDataLength*

The size of the buffer containing the additional data.
 Invalid for German fiscal element (TSE) model "TSE1", in which case a value of 0 for this parameter is recommended.

[OUT] *logTime*

Pointer to the buffer that receives the value that corresponds to the date and time the log was created. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.
 Cannot be a null pointer.

[OUT] *signatureValue*

Pointer to the buffer that receives the raw binary form of the serial number of the TSE.
 The memory block needed to receive the serial number is allocated internally by this function and can be deallocated using the TSEMemDealloc.
 Cannot be a null pointer.

[OUT] *signatureValueLength*

Pointer that receives the size of the memory block allocated for *signatureValue*.
 Cannot be a null pointer.

[OUT] *signatureCounter*

Pointer to the buffer that receives the current signature counter value.
 Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportDataFilteredByTransactionNumberAndClientId

Exports the log messages associated with the specified client ID and transaction number.

- ❑ Acquires the specified target log saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ This function internally executes `FinalizeExport` if target data is successfully exported.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int ExportDataFilteredByTransactionNumberAndClientId(
    void                *tseHandle,
    unsigned long int    transactionNumber,
    unsigned char        *clientId,
    unsigned long int    clientIdLength,
    unsigned char        **exportedData,
    unsigned long int    *exportedDataLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *transactionNumber*

Transaction number of the log to be exported.

[IN] *clientId*

Pointer to the buffer that contains the client ID of which the log to be exported is associated. Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID.

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for `exportedData`.
Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.
Otherwise, the function returns an Error code.

ExportDataFilteredByTransactionNumber

Exports the log messages associated with the specified transaction number.

- ❑ Acquires the specified target log saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ This function internally executes `FinalizeExport` if target data is successfully exported.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int ExportDataFilteredByTransactionNumber (
    void                *tseHandle,
    unsigned long int    transactionNumber,
    unsigned char        **exportedData,
    unsigned long int    *exportedDataLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *transactionNumber*

Transaction number of the log to be exported.

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for `exportedData`.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportDataFilteredByTransactionNumberIntervalAndClientId

Exports the log messages that correspond to the specified client ID and transaction number interval.

Only the log messages in the specified transaction interval that also correspond to the specified client ID is exported.

- ❑ Acquires the specified target log saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ This function internally executes `FinalizeExport` if target data is successfully exported.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int
ExportDataFilteredByTransactionNumberIntervalAndClientId(
    void                *tseHandle,
    unsigned long int   startTransactionNumber,
    unsigned long int   endTransactionNumber,
    unsigned char       *clientId,
    unsigned long int   clientIdLength,
    unsigned char       **exportedData,
    unsigned long int   *exportedDataLength);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **startTransactionNumber**

Starting number of the transaction number interval of the relevant logs.

[IN] **endTransactionNumber**

Ending number of the transaction number interval of the relevant logs.
`startTransactionNumber` must be \leq `endTransactionNumber`

[IN] **clientId**

Pointer to the buffer that contains the client ID of which the log to be exported is associated.
 Cannot be a null pointer.

[IN] **clientIdLength**

The size of the buffer containing the client ID.

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the TSEMemDealloc.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for *exportedData*.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportDataFilteredByTransactionNumberInterval

Exports the log messages that correspond to the specified transaction number interval.

- ❑ Acquires the specified target log saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ This function internally executes `FinalizeExport` if target data is successfully exported.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int ExportDataFilteredByTransactionNumberInterval (
    void                *tseHandle,
    unsigned long int    startTransactionNumber,
    unsigned long int    endTransactionNumber,
    unsigned char        **exportedData,
    unsigned long int    *exportedDataLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *startTransactionNumber*

Starting number of the transaction number interval of the relevant logs.

[IN] *endTransactionNumber*

Ending number of the transaction number interval of the relevant logs.
`startTransactionNumber` must be \leq `endTransactionNumber`

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for `exportedData`.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportDataFilteredByPeriodOfTimeAndClientId

Exports the log messages that correspond to the specified client ID and date and time range.

Only the log messages in the specified date and time range that also correspond to the specified client ID is exported.

- ❑ Acquires the specified target log saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ This function internally executes `FinalizeExport` if target data is successfully exported.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int ExportDataFilteredByPeriodOfTimeAndClientId(
    void                *tseHandle,
    unsigned long int    startDate,
    unsigned long int    endDate,
    unsigned char        *clientId,
    unsigned long int    clientIdLength,
    unsigned char        **exportedData,
    unsigned long int    *exportedDataLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *startDate*

The value that corresponds to the starting date and time of the relevant logs. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.

[IN] *endDate*

The value that corresponds to the ending date and time of the relevant logs. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.

[IN] *clientId*

Pointer to the buffer that contains the client ID of which the log to be exported is associated. Cannot be a null pointer.

[IN] *clientIdLength*

The size of the buffer containing the client ID.

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the TSEMemDealloc.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for *exportedData*.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportDataFilteredByPeriodOfTime

Exports the log messages that correspond to the specified date and time range.

- ❑ Acquires the specified target log saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ This function internally executes `FinalizeExport` if target data is successfully exported.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int ExportDataFilteredByPeriodOfTime (
    void                *tseHandle,
    unsigned long int    startDate,
    unsigned long int    endDate,
    unsigned char        **exportedData,
    unsigned long int    *exportedDataLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *startDate*

The value that corresponds to the starting date and time of the relevant logs. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.

[IN] *endDate*

The value that corresponds to the ending date and time of the relevant logs. The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for `exportedData`.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ExportData

Exports all log data.

- ❑ Acquires all log data saved in the TSE in TAR file format.
- ❑ Before starting exporting, the TSE time must be updated by executing `UpdateTime`.
- ❑ Exporting cannot be restarted while already in export status.
- ❑ Once all log data is successfully exported, the `FinalizeExport` must be explicitly executed.
- ❑ If data is not received properly while exporting due to an error such as disconnected communication, the exported data may be invalid, `CancelExport` must be executed to start exporting again.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int ExportData(
    void          *tseHandle,
    unsigned char **exportedData,
    unsigned long int *exportedDataLength);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *exportedData*

Pointer to the buffer that receives the exported data in binary form, which can be saved as a single TAR file.

The memory block needed to receive the exported data is allocated internally by this function and can be deallocated using the `TSEMemDealloc`.

Cannot be a null pointer.

[OUT] *exportedDataLength*

Pointer that receives the size of the memory block allocated for `exportedData`.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

FinalizeExport

Ends exporting.

- ❑ This function clears the export status.
- ❑ If `deleteStoredData`, is set to true, all the log data in the TSE is deleted.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int FinalizeExport(
    void          *tseHandle,
    bool          deleteStoredData);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] *deleteStoredData*

Indicate whether to delete all log data in the TSE.

true: delete. false: do not delete

This parameter (more particularly when set to true) is only valid when an `ExportData` has just been successfully executed.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

CancelExport

Cancels exporting.

- ❑ This function clears the export status.
- ❑ To use this function, the client must be logged in with Admin privileges.

Syntax

```
short int CancelExport(  
    void *tseHandle);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

ChangePuk

Changes the PUK.

- ❑ If PUK is blocked, it cannot be recovered. The only way to recover is to replace the TSE.

Syntax

```
short int ChangePuk(
    void                *tseHandle,
    unsigned char       *oldPuk,
    unsigned long int    oldPukLength,
    unsigned char       *newPuk,
    unsigned long int    newPukLength,
    unsigned char       *secretKey,
    unsigned long int    secretKeyLength,
    short int           *remainingRetries);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **oldPuk**

Pointer to the buffer that contains the current PUK.
Cannot be a null pointer.

[IN] **oldPukLength**

The size of the buffer containing the current PUK.

[IN] **newPuk**

Pointer to the buffer that contains the new PUK to be set.
Cannot be a null pointer.

[IN] **newPukLength**

The size of the buffer containing the new PUK.

[IN] **secretKey**

Pointer to the buffer that contains the secret key used to calculate the hash value for authentication.
Cannot be a null pointer.

[IN] **secretKeyLength**

The size of the buffer containing the secret key.

[OUT] *remainingRetries*

Pointer to receive the number of input PUK retries remaining.

PIN/PUK authentication can fail up to 3 times. This will be -1 when PUK authentication is successful or when an error other than authentication failure is encountered.

Reduces by 1 on each PUK authentication failure.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

UnblockUser

Unlocks the PIN for Admin or TimeAdmin privileges.

- ❑ If PUK is blocked, it cannot be recovered. The only way to recover is to replace the TSE.

Syntax

```
short int UnblockUser(
    void                *tseHandle,
    unsigned char       *userId,
    unsigned long int   userIdLength,
    unsigned char       *puk,
    unsigned long int   pukLength,
    unsigned char       *newPin,
    unsigned long int   newPinLength,
    unsigned char       *secretKey,
    unsigned long int   secretKeyLength,
    short int           *remainingRetries);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **userId**

Pointer to the buffer that contains the user ID of the PIN to be unlocked.

If the target is to unblock the Admin PIN, then the user ID must be "Administrator".

If the target is to unblock the TimeAdmin PIN, then the user ID must be a TimeAdmin client ID.

Cannot be a null pointer

[IN] **userIdLength**

The size of the buffer containing the user ID.

[IN] **puk**

Pointer to the buffer that contains the PUK.

Cannot be a null pointer.

[IN] **pukLength**

The size of the buffer containing the PUK.

[IN] **newPin**

Pointer to the buffer that contains the new PIN to be set.

Cannot be a null pointer.

[IN] *newPinLength*

The size of the buffer containing the new PIN.

[IN] *secretKey*

Pointer to the buffer that contains the secret key used to calculate the hash value for authentication.

Cannot be a null pointer.

[IN] *secretKeyLength*

The size of the buffer containing the secret key.

[OUT] *remainingRetries*

Pointer to receive the number of input PUK retries remaining.

PIN/PUK authentication can fail up to 3 times. This will be -1 when PUK authentication is successful or when an error other than authentication failure is encountered.

Reduces by 1 on each PUK authentication failure.

Cannot be a null pointer.

Return value

The function return `EXECUTION_OK` (0) if executed successfully.

Otherwise, the function returns an Error code.

TSEMemDealloc

Deallocates a memory block allocated by the DLL for the result of a TSE operation.

Syntax

```
void      TSEMemDealloc (
    void          *tseHandle,
    void          *memoryAddress);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *memoryAddress*

Pointer to the memory block to be deallocated.
Cannot be a null pointer.

TSEMemDeallocTseInfoObject

Deallocates the memory blocks allocated by the DLL for an EPSON_TSE_INFO instance.

- ❑ Deallocates the blocks used for the serialNumber, tsePublicKey, rawTseInfo of the specified EPSON_TSE_INFO instance.

Syntax

```
void      TSEMemDeallocTseInfoObject (
    void      *tseHandle,
    EPSON_TSE_INFO *tseInfo);
```

Parameter

[IN] *tseHandle*

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[OUT] *tseInfo*

Pointer to the EPSON_TSE_INFO instance.
Cannot be a null pointer.

TSESetLog

Sets the DLL's logging method and level.

- ❑ Logging is enabled by default in the DLL (log level set to TSELIBCLOGVLINFO).
Default log method is set to TSELIBCLOGFILE (log on file).
- ❑ If the log method is set to the logging on a file (e.g. TSELIBCLOGFILE, etc.), then a log file will be created for the TSE connection referenced by the `tseHandle`. The filename of the log will be `[hostid].[tseid].log`.
- ❑ The DLL also creates an additional log file named EPSONTSELIBCMULTI.log. The log level for this file is always TSELIBCLOGVLTRACE and cannot be changed.
- ❑ All log files will be placed in %PROGRAMDATA%\Epson\TSE\CLIBLogs\.

Syntax

```
void          TSESetLog (
    void          *tseHandle,
    unsigned short int  logLevel,
    unsigned short int  logMethod);
```

Parameter

[IN] **tseHandle**

The descriptor (of type `void*`) that identifies the target TSE connection. The descriptor is created by `TSEConnect()`.

[IN] **logLevel**

The log level.

Value	Description
TSELIBCLOGVLDISABLE	Disable logging.
TSELIBCLOGVLERROR	Runtime errors.
TSELIBCLOGVLWARNING	Warnings. runtime situations that are undesirable or unexpected, but not necessarily "wrong"
TSELIBCLOGVLINFO	Information related to DLL.
TSELIBCLOGVLTRACE	Function traces.
TSELIBCLOGVLVERBOSE	All including bytes sent and received by the DLL and memory blocks allocated by the DLL.

[IN] *logMethod*

The log method.

Value	Description
TSELIBCLOGDEBUGGER	Output logs on Debugger (DebugView)
TSELIBCLOGFILE	Output logs on a file
TSELIBCLOGSTDOUT	Output logs on standard output (Console)
TSELIBCLOGDEBUGGERANDFILE	Output logs on Debugger (DebugView), and Output logs on a file
TSELIBCLOGDEBUGGERANDSTD OUT	Output logs on Debugger (DebugView), and Output logs on standard output (Console)
TSELIBCLOGFILEANDSTDOUT	Output logs on standard output (Console), and Output logs on a file
TSELIBCLOGALL	Output logs on Debugger (DebugView), and Output logs on standard output (Console), and Output logs on a file

TSEGetErrorDescription

Acquires the description of the error that corresponds to the specified Error code.

Syntax

```
void TSEGetErrorDescription(  
    short int      errorCode,  
    short int      language,  
    unsigned char  errorDescription[100]);
```

Parameter

[IN] *errorCode*

The error code.

[IN] *language*

Language of the description of the error.

Value	Language
0	English
1	German

[OUT] *errorDescription*

Buffer to receive the description of the error. The buffer will contain a null terminated string, that requires up to 100 bytes of memory.

EPSON_TSE_INFO

```
#define NULLCHAR 1
typedef struct
{
    unsigned long int    softwareVersion;
    unsigned long int    hardwareVersion;
    unsigned long int    tseCapacity;
    unsigned long int    tseCurrentSize;
    unsigned long int    timeUntilNextSelfTest;
    unsigned long int    maxUpdateDelay;
    unsigned long int    startedTransactions;
    unsigned long int    maxStartedTransactions;
    unsigned long int    createdSignatures;
    unsigned long int    remainingSignatures;
    unsigned long int    maxSignatures;
    unsigned long int    registeredClients;
    unsigned long int    maxRegisteredClients;
    unsigned long int    tarExportSize;
    unsigned long int    certificateExpirationDate;
    unsigned long int    lastExportExecutedDate;
    unsigned char        certificateExpirationDateStr[30 + NULLCHAR];
    unsigned char        lastExportExecutedDateStr[30 + NULLCHAR];
    unsigned char        vendorType[100 + NULLCHAR];
    unsigned char        tseInitializationState[100 + NULLCHAR];
    unsigned char        tseDescription[128 + NULLCHAR];
    unsigned char        signatureAlgorithm[100 + NULLCHAR];
    unsigned char        cdcId[18 + NULLCHAR];
    unsigned char        cdcHash[64 + NULLCHAR];
    unsigned char        *serialNumber = NULL;
    unsigned long int    serialNumberLength = 0;
    unsigned char        *tsePublicKey = NULL;
    unsigned long int    tsePublicKeyLength = 0;
    unsigned char        *rawTseInfo = NULL;
    unsigned long int    rawTseInfoLength = 0;
    bool                hasValidTime;
    bool                hasPassedSelfTest;
    bool                isTransactionInProgress;
    bool                isTSEUnlocked;
    bool                isExportEnabledIfCspTestFails;
} EPSON_TSE_INFO;
```

Definitions

Parameter	Description
softwareVersion	The German fiscal element (TSE)'s software version
hardwareVersion	The German fiscal element (TSE)'s hardware version
tseCapacity	German fiscal element (TSE) capacity <ul style="list-style-type: none"> Valid range: range of unsigned long int type Units: blocks. 1Block = 512 Byte
tseCurrentSize	German fiscal element (TSE) size used <ul style="list-style-type: none"> Valid range: range of unsigned long int type Units: blocks. 1Block = 512 Byte. Returns a correct value only when the German fiscal element (TSE) has been unlocked, and the self-test has been done, otherwise 0 is returned.
timeUntilNextSelfTest	The timeout in seconds after which the next self-test must be run. <ul style="list-style-type: none"> Valid range: range for "unsigned int" type. If this reaches 0, all following commands will fail until the self-test is executed again. This time will only change after the first time update on the TSE.
maxUpdateDelay	Interval (in seconds) after which a started transaction must have received an update in case new data is available on the cash register. This is currently set to 45 seconds according to MAX_UPDATE_DELAY from [BSI-TR-03116-5].
startedTransactions	Number of transactions that have not been finished, yet. If this equals Max Started Transactions, no new transactions can be started until at least one transaction has been finished. <ul style="list-style-type: none"> Valid range: 0 to 512 Returns the correct value only when German fiscal element (TSE) is unlocked
maxStartedTransactions	Maximum number of started transactions. i.e. amount of transactions that can be started in parallel. <ul style="list-style-type: none"> Maximum value 512
createdSignatures	Number of signatures that have been created with this German fiscal element (TSE). <ul style="list-style-type: none"> Valid range: 0 to 10000000
remainingSignatures	Remaining amount of signatures <ul style="list-style-type: none"> Valid range: 0 to 10000000
maxSignatures	Maximum number of signatures that can be created with this German fiscal element (TSE). <ul style="list-style-type: none"> Maximum value 10000000
registeredClients	The number of clients that is currently registered clients. <ul style="list-style-type: none"> Valid range: 0 to 100 However, as one client ID is used by the printer, up to 99 client IDs can be registered by the user.
maxRegisteredClients	Maximum number of clients that can be registered. <ul style="list-style-type: none"> Maximum value 100 However, as one client ID is used by the printer, up to 99 client IDs can be registered by the user.
tarExportSize	Size of the whole German fiscal element (TSE) Store in bytes, if exported. <ul style="list-style-type: none"> Valid range: range of unsigned long int type Units: bytes

Parameter	Description
	<ul style="list-style-type: none"> Returns a correct value only when the German fiscal element (TSE) has been unlocked, and the self-test has been done, otherwise 0 is returned.
certificateExpirationDate	<p>Value that corresponds to the timestamp after which the certificate of this German fiscal element (TSE) will be invalid. As the German fiscal element (TSE) will not be usable afterwards, all data must have been properly exported before this date.</p> <ul style="list-style-type: none"> The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.
lastExportExecutedDate	<p>Value that corresponds to the date and time of the last export</p> <p>* Date and time that FinalizeExport was last executed successfully.</p> <ul style="list-style-type: none"> The value is in integral form that indicates the number of seconds (not counting leap seconds) since 00:00, January 1, 1970 UTC.
certificateExpirationDateStr	<p>Same value as that of certificateExpirationDate, but in null terminated string format.</p> <ul style="list-style-type: none"> UTC specification <p>Format is ISO 8601 extended format</p> <p>- YYYY-MM-DDThh:mm:ssZ</p>
lastExportExecutedDateStr	<p>Same value as that of lastExportExecutedDate, but in null-terminated string format.</p> <ul style="list-style-type: none"> UTC specification <p>Format is ISO 8601 extended format</p> <p>- YYYY-MM-DDThh:mm:ssZ</p>
vendorType	A null terminated string that contains the German fiscal element (TSE) model name
tseInitializationState	<p>A null terminated string that contains the German fiscal element (TSE) initialization status</p> <ul style="list-style-type: none"> "UNINITIALIZED" <p>: uninitialized status</p> <ul style="list-style-type: none"> "INITIALIZED" <p>: initialized status</p> <ul style="list-style-type: none"> "DECOMMISSIONED" <p>: cannot be used</p>
tseDescription	A null terminated ASCII string containing the certificate ID issued by BSI to prove TR-03153 compliance.
signatureAlgorithm	A null terminated string containing the signature algorithm that is used by the German fiscal element (TSE).
cdclId	A null terminated string that contains the unique identifier created by Epson to track a specific German fiscal element (TSE).
cdcHash	A null terminated string that contains the hash to confirm the integrity of the data.
serialNumber	Pointer to a buffer that receives the binary form of the serial number of the TSE.
serialNumberLength	The size of the buffer that contains binary of the serial number.
tsePublicKey	<p>Pointer to a buffer that receives the binary form of the public key that belongs to the private key generating signatures.</p> <p>This key can be used to verify all signatures created by the TSE.</p>
tsePublicKeyLength	The size of the buffer that contains binary form of the public key.
rawTseInfo	Pointer to a buffer that receives the TSE Information in raw JSON string format.

Parameter	Description
rawTseInfoLength	The size of the buffer that receives the TSE Information in raw JSON string format.
hasValidTime	Whether a valid time is set in German fiscal element (TSE) (whether UpdateTime has been executed) true: valid time set false: valid time not set • A transaction can be started only if a valid time has been set.
hasPassedSelfTest	Whether the German fiscal element (TSE) passed its self-test. true: German fiscal element (TSE) self-test has passed false: German fiscal element (TSE) self-test has not passed
isTransactionInProgress	Whether a transaction is currently in progress. true: a transaction is currently in progress false: a transaction is not currently in progress
isTSEUnlocked	German fiscal element (TSE) lock status * Transactions and export functions are not available when the German fiscal element (TSE) is locked. true: German fiscal element (TSE) is unlocked false: German fiscal element (TSE) is locked • This will only be true, if the German fiscal element (TSE) has been unlocked and the self-test has been successful.
isExportEnabledIfCspTestFails	Whether data export is enabled if the CSP test failed. true: data export is enabled if the CSP test failed false: data export is disabled if the CSP test failed

EPSON_TSE_SMART_INFO

```
#define NULLCHAR 1
typedef struct
{
    unsigned char        spareBlockStatus_healthStatus[64 + NULLCHAR];
    unsigned long int    spareBlockStatus_remainingSpareBlocks;
    unsigned char        dataIntegrity_healthStatus[64 + NULLCHAR];
    unsigned long int    dataIntegrity_uncorrectableECCErrors;
    unsigned char        eraseLifetimeStatus_healthStatus[64 + NULLCHAR];
    unsigned long int    eraseLifetimeStatus_remainingEraseCounts;
    unsigned long int    remainingTenYearsDataRetention;
    unsigned char        tseHealth[64 + NULLCHAR];
    bool                 isReplacementNeeded;
} EPSON_TSE_SMART_INFO;
```

Definitions

Parameter	Description
spareBlockStatus_healthStatus	<p>SpareBlockStatus: HealthStatus</p> <p>Available values are as follows:</p> <ul style="list-style-type: none"> • "DEAD" when the value of remainingSpareBlocks is less than 25 • "WARNING" when the value of remainingSpareBlocks is less than 30 • "PASS" when the value of remainingSpareBlocks is other than the above • Returns a correct value only when the self-test has been done. <p>Returns "UNKNOWN" except for that.</p>
spareBlockStatus_remainingSpareBlocks	<p>Percentage of remaining spare blocks.</p> <ul style="list-style-type: none"> • Valid range: 0 to 100 • Returns a correct value only when the self-test has been done. <p>Returns 100 except for that.</p>
dataIntegrity_healthStatus	<p>DataIntegrity: HealthStatus.</p> <p>Available values are as follows:</p> <ul style="list-style-type: none"> • "DEAD" when the value of uncorrectableECCErrors is 1 or higher • "PASS" when the value of uncorrectableECCErrors is 0 • Returns a correct value only when the self-test has been done. <p>Returns "UNKNOWN" except for that.</p>
dataIntegrity_uncorrectableECCErrors	<p>Shows the number of uncorrectable(unrecoverable) ECC errors that occurred on read commands.</p> <ul style="list-style-type: none"> • Valid range: range of unsigned short int type • Returns a correct value only when the self-test has been done. <p>Returns 0 except for that.</p>
eraseLifetimeStatus_healthStatus	<p>EraseLifetimeStatus: HealthStatus</p> <p>Available values are as follows:</p> <ul style="list-style-type: none"> • "DEAD" when the value of remainingEraseCounts is less than 2 • "WARNING" when the value of remainingEraseCounts is less than 10 • "PASS" when the value of remainingEraseCounts is other than the above • Returns a correct value only when the self-test has been done. <p>Returns "UNKNOWN" except for that.</p>
eraseLifetimeStatus_remainingEraseCounts	<p>Percentage of remaining erase counts.</p> <ul style="list-style-type: none"> • Valid range: 0 to 100 • Returns a correct value only when the self-test has been done. <p>Returns 100 except for that.</p>
remainingTenYearsDataRetention	<p>Percentage of remaining erase counts until the ten year data retention can not be guaranteed</p>

Parameter	Description
	<p>anymore.</p> <ul style="list-style-type: none"> Valid range: 0 to 100 Returns a correct value only when the self-test has been done. <p>Returns 0 except for that.</p>
tseHealth	<p>Available values are as follows:</p> <ul style="list-style-type: none"> "DEAD" when "healthStatus" in any one of "spareBlockStatus", "eraseLifetimeStatus", or "dataIntegrity" is "DEAD". "WARNING" when "healthStatus" in each of "spareBlockStatus", "eraseLifetimeStatus", and "dataIntegrity" is not "DEAD" and "healthStatus" in any one of the above is "WARNING". PASS" when "healthStatus" in "spareBlockStatus", "eraseLifetimeStatus", and "dataIntegrity" is all "PASS". Returns a correct value only when the self-test has been done. <p>Returns "UNKNOWN" except for that.</p>
isReplacementNeeded	<p>Whether the German fiscal element (TSE) should be replaced with a new one based on the current flash health.</p> <p>true: German fiscal element (TSE) should be replaced</p> <p>false: German fiscal element (TSE) does not need to be replaced</p> <p>* Set to true when tseHealth is "DEAD".</p> <ul style="list-style-type: none"> Returns a correct value only when the self-test has been done. <p>Returns false except for that.</p>

Error code

Error code	Description
EXECUTION_OK	Execution has successfully completed.
EXECUTION_COMPLETED_BUT_WITH_ERROR_ALLOCATING_FOR_RESULT	Operation on the TSE is completed however DLL failed to allocate memory for the result.
EXECUTION_COMPLETED_BUT_WITH_ERROR_DECODING_RESULT	Operation on the TSE is completed however DLL failed to decode the result.
EXECUTION_COMPLETED_BUT_WITH_UNFINALIZED_EXPORT	Operation on the TSE is completed however DLL failed to end the export (FinalizeExport).
JSON_ERROR_NOT_ENOUGH_CONTENTS	Inadequate JSON message contents (missing parameter, etc.).
JSON_ERROR_UNEXPECTED_PARAM_TYPE	Incorrect JSON message parameter data type.
JSON_ERROR_OVER_DATA_SIZE	JSON message size exceeds the maximum value supported (64 KB).
JSON_ERROR_INVALID_PARAMETER_RANGE	JSON message parameters are outside their valid range.
JSON_ERROR_INVALID_PARAMETER	Invalid parameter.
JSON_ERROR_INVALID_TIME_FORMAT	Invalid (not supported) timestamp format for a specified JSON message parameter.
JSON_ERROR_UNEXPECTED_STORAGE_TYPE	Unexpected (not supported) "storage" "type" specified by a JSON message parameter.
JSON_ERROR_UNEXPECTED_STORAGE_VENDOR	Unexpected (not supported) "storage" "vendor" specified by a JSON message parameter.
JSON_ERROR_UNEXPECTED_FUNCTION	Unexpected "function" specified by a JSON message parameter.
JSON_ERROR_UNEXPECTED_COMPRESSION_TYPE	Unexpected (not supported) "compress" "type" specified by a JSON message parameter.
JSON_ERROR_WRONG_JSON_FORMAT	JSON message is not in JSON format.
OTHER_ERROR_UNKNOWN_STORAGE	Inserted storage is not a supported Germany Fiscal Element (TSE).
OTHER_ERROR_NO_STORAGE_FOUND	No valid storage is inserted.
OTHER_ERROR_UPDATE_SYSTIME_FAIL	Failed to update the printer time.
OTHER_ERROR_INVALID_ADMIN_USER_ID	A user ID with Admin privileges besides "Administrator" was specified.
OTHER_ERROR_HOST_AUTHENTICATION_FAILED	Failed to authenticate the host. Incorrect hash value.
OTHER_ERROR_UNAUTHENTICATED_TIME_ADMIN_USER	A request was received from an unauthenticated (not logged-in) user with TimeAdmin privileges.
OTHER_ERROR_UNAUTHENTICATED_ADMIN_USER	A request was received from an unauthenticated (not logged-in) user with Admin privileges.
OTHER_ERROR_UNAUTHENTICATED_USER	A request was received from a user not authenticated with Admin or TimeAdmin privileges.
OTHER_ERROR_UNAUTHENTICATED_HOST	A request was received from an unauthenticated (not logged-in) host.
OTHER_ERROR_CURRENTLY_EXPORTING	Another request was received while in export status.
OTHER_ERROR_NO_EXPORT_STARTED	A request was received that can only be executed in export status while not in export status.
OTHER_ERROR_CURRENTLY_EXPORTING_USER	The user ID who is currently exporting has logged out.
OTHER_ERROR_NO_TIME_SET_BEFORE_EXPORT	Germany Fiscal Element (TSE) time was not updated before starting the export.
OTHER_ERROR_ALREADY_ALL_DATA_EXPORTED	An export data acquisition request was received when all export data had been acquired.
OTHER_ERROR_ALL_DATA_NOT_EXPORTED_YET	An export end request was received when all export data had not been acquired.

Error code	Description
OTHER_ERROR_TSE_ALREADY_SET_UP	Another set-up request was received for a Germany Fiscal Element (TSE) that is already set-up.
OTHER_ERROR_SECRETKEY_REGISTRATION_FAILED	Failed to register the secret key.
OTHER_ERROR_PARAMETER_MISMATCH	Incorrect parameters entered to specify the export range.
OTHER_ERROR_TOO_MANY_TIME_ADMIN_USER_LOGGED_IN	The number of users (clients) logged in with TimeAdmin privileges exceeds the upper limit.
OTHER_ERROR_TOO_MANY_CHALLENGE_REQUESTED	The number of users requesting the challenge exceeds the upper limit.
OTHER_ERROR_NO_MEMORY	Memory overflow error. This does not normally occur.
OTHER_ERROR_FATAL	Fatal error due to an unexpected event (mainly at the Germany Fiscal Element (TSE) side).
OTHER_ERROR_CANNOT_GET_TSE_LIST	Cannot retrieve list of TSE devices.
TSE1_ERROR_INVALID_PARAMETER	Invalid input parameter.
TSE1_ERROR_NO_TSE	No Germany Fiscal Element (TSE) was found at the provided path.
TSE1_ERROR_IO	No Germany Fiscal Element (TSE) detected (inserted) when a command request arrived.
TSE1_ERROR_TIMEOUT	The operation timed out.
TSE1_ERROR_OUTOFMEM	A location was accessed that is outside the memory.
TSE1_ERROR_INVALID_RESPONSE	No correct response from Germany Fiscal Element (TSE).
TSE1_ERROR_STORE_FULL_INTERNAL	Germany Fiscal Element (TSE) is internally full.
TSE1_ERROR_RESPONSE_MISSING	This happens if two commands are sent at the same time or if a command is not allowed in the current state.
TSE1_ERROR_EXPORT_NOT_INITIALIZED	Germany Fiscal Element (TSE) Store not initialized.
TSE1_ERROR_EXPORT_FAILED	Export Failed.
TSE1_ERROR_POWER_CYCLE_DETECTED	A power cycle occurred during command execution.
TSE1_ERROR_FIRMWARE_UPDATE_NOT_APPLIED	The firmware update was not properly applied.
TSE1_ERROR_FROM_TSE_FIRST	Lowest error code that might be raised from the Germany Fiscal Element (TSE).
TSE1_ERROR_UNKNOWN	Unspecified, internal processing error.
TSE1_ERROR_NO_TIME_SET	Time not set.
TSE1_ERROR_NO_TRANSACTION_IN_PROGRESS	No transaction in progress.
TSE1_ERROR_INVALID_CMD_SYNTAX	Wrong command length.
TSE1_ERROR_NOT_ENOUGH_DATA_WRITTEN	Not enough data written during transaction.
TSE1_ERROR_TSE_INVALID_PARAMETER	Invalid Parameter.
TSE1_ERROR_TRANSACTION_NOT_STARTED	Given transaction is not started.
TSE1_ERROR_MAX_PARALLEL_TRANSACTIONS	Maximum parallel transactions reached.
TSE1_ERROR_CERTIFICATE_EXPIRED	Certificate expired.
TSE1_ERROR_NO_LAST_TRANSACTION	No last transaction to fetch.
TSE1_ERROR_CMD_NOT_ALLOWED	Command not allowed in current state.
TSE1_ERROR_TRANSACTION_SIGNATURES_EXCEEDED	Signatures exceeded.
TSE1_ERROR_NOT_AUTHORIZED	Not authorized.
TSE1_ERROR_MAX_REGISTERED_CLIENTS_REACHED	Maximum registered clients reached.

Error code	Description
TSE1_ERROR_CLIENT_NOT_REGISTERED	Client not registered.
TSE1_ERROR_EXPORT_UNACKNOWLEDGED_DATA	Failed to delete, data not completely exported.
TSE1_ERROR_CLIENT_HAS_UNFINISHED_TRANSACTIONS	Failed to deregister, client has unfinished transactions.
TSE1_ERROR_TSE_HAS_UNFINISHED_TRANSACTIONS	Failed to decommission, Germany Fiscal Element (TSE) has unfinished transactions.
TSE1_ERROR_TSE_NO_RESPONSE_TO_FETCH	Wrong state, there is no response to fetch.
TSE1_ERROR_NOT_ALLOWED_EXPORT_IN_PROGRESS	Wrong state, ongoing Filtered Export must be finished before this command is allowed.
TSE1_ERROR_STORE_FULL	Operation failed, not enough remaining capacity in Germany Fiscal Element (TSE) Store.
TSE1_ERROR_WRONG_STATE_NEEDS_PUK_CHANGE	Wrong state, changed PUK required.
TSE1_ERROR_WRONG_STATE_NEEDS_PIN_CHANGE	Wrong state, changed PIN required.
TSE1_ERROR_WRONG_STATE_NEEDS_TSE_UNLOCK	Wrong state, unlocked Germany Fiscal Element (TSE) required.
TSE1_ERROR_WRONG_STATE_NEEDS_SELF_TEST	Wrong state, self test must be run first.
TSE1_ERROR_WRONG_STATE_NEEDS_SELF_TEST_PASSED	Wrong state, passed self test required.
TSE1_ERROR_FWU_INTEGRITY_FAILURE	Firmware Update: Integrity check failed.
TSE1_ERROR_FWU_DECRYPTION_FAILURE	Firmware Update: Decryption failed.
TSE1_ERROR_FWU_WRONG_FORMAT	Firmware Update: Wrong format.
TSE1_ERROR_FWU_INTERNAL_ERROR	Firmware Update: Internal error.
TSE1_ERROR_FWU_DOWNGRADE_PROHIBITED	Firmware Update: downgrade prohibited.
TSE1_ERROR_TSE_ALREADY_INITIALIZED	Germany Fiscal Element (TSE) already initialized.
TSE1_ERROR_TSE_DECOMMISSIONED	Germany Fiscal Element (TSE) decommissioned.
TSE1_ERROR_TSE_NOT_INITIALIZED	Germany Fiscal Element (TSE) not initialized.
TSE1_ERROR_AUTHENTICATION_FAILED	Authentication failed.
TSE1_ERROR_AUTHENTICATION_PIN_BLOCKED	PIN is blocked.
TSE1_ERROR_AUTHENTICATION_USER_NOT_LOGGED_IN	Given user is not authenticated.
TSE1_ERROR_SELF_TEST_FAILED_FW	Self test of FW failed.
TSE1_ERROR_SELF_TEST_FAILED_CSP	Self test of CSP failed.
TSE1_ERROR_SELF_TEST_FAILED_RNG	Self test of RNG failed.
TSE1_ERROR_FWU_BASE_FW_ERROR	Firmware Update: Base FW update error.
TSE1_ERROR_FWU_FWEXT_ERROR	Firmware Update: FW Extension update error.
TSE1_ERROR_FWU_CSP_ERROR	Firmware Update: CSP update error.
TSE1_ERROR_EXPORT_NONE_IN_PROGRESS	Filtered Export: no export in progress.
TSE1_ERROR_EXPORT_RETRY	Filtered Export: no new data, keep polling.
TSE1_ERROR_EXPORT_NO_DATA_AVAILABLE	Filtered Export: no matching entries, export would be empty.
TSE1_ERROR_CMD_NOT_FOUND	Command not found.
TSE1_ERROR_SIG_ERROR	Signature creation error.
TSE1_ERROR_FROM_TSE_LAST	Highest error code that might be raised from the Germany Fiscal Element (TSE).
TSE1_ERROR_RAISED_FROM_WORM_CARD	Other Errors that raised from the Germany Fiscal Element (TSE).

Error code	Description
DEVICE_NO_CONNECTION	No connection has been established with the TSE.
DEVICE_CONNECTION_ERROR	Connection with the TSE has failed.
OTHER_ERROR_INVALID_POINTER	Pointer passed to the API function is invalid.
OTHER_ERROR_ENCODING_FAILED	Encoding data failed.
OTHER_ERROR	Unknown Error.