TENeT e-Right/ e-Money Transaction API Specification

TEF950-S001-01.00.00/en

2005/03

T-Engine Forum

TENeT e-Right/e-Money Transaction API Specification

Scope

This document defines the TENeT e-right/e-money transaction API, which is a Java API designed to conduct such operations as the reading/creation of electronic rights and money, and such transactions as buy and sell, exchange, etc., by using TENeT-enabled IC cards.

Normative References

e^{2}TP messaging API:

> e^{2}TP message specification, T-Engine Forum, 2005
> Gosling, J., Joy, B., Steele, G., and Bracha, G.: The Java Language Specification, Sun Microsystems, 2000.

VTS API:

> TENeT message specification, T-Engine Forum, 2005
> Gosling, J., Joy, B., Steele, G., and Bracha, G.: The Java Language Specification, Sun Microsystems, 2000.

Introduction

The TENeT e-right/e-money transaction API (org.t_engine.tenet.vts) is a Java API that is designed to conduct operations and transactions of electronic rights and money using TENeT-enabled IC cards.

The representative functions provided by this API are given below, among others:

- Authentication on IC cards (`Participant#getAgent()`)
- Acquisition of folder table (`Agent#getFolders()`)
- Creation and deletion of folders (`Agent#createdFolder()/deletedFolder()`)
- Acquisition of e-right/e-money table (`stored in folders`)(`Folder#iterator()`)
- Creation of e-rights/e-money (`Folder#createVoucher()`)
- Deletion of e-rights/e-money (`StoredVoucher#remove()`)
- Transfer/exchange of e-rights/e-money (`StoredVoucher#transfer()/exchange()`)
- Recovery of suspended session (`Session#recover()`)

As shown above, most of the functions provided by the TENeT-enabled IC card (excluding part of the operational functions, such as the backup/recovery of contents of cards, etc.) are available through this API alone. In other words, use of this API allows the application developer to make use of the diverse types of functions provided by the TENeT-enabled IC card, without having to directly take into account the transmission/reception of e^{2}TP messages.

On this API, the transactions (transfer and exchange) of e-rights/e-money are carried out asynchronously. That is, when StoredVoucher#transfer()/exchange() is invoked, it returns immediately, without waiting for its result. Thereafter, the listeners (see Agent#setListener()) registered at Agents of both parties to the transaction are called up successively, according as the transaction advances.

An execution of a transaction proceeds as follows, where the AP of the side proposing a transaction is indicated as 'A', and the AP of the side accepting the transaction, as 'B'. Moreover, the listener registered at 'A' is termed lis_A, and the listener registered at 'B', lis_B.

1. 'A' initiates a transaction (StoredVoucher#exchange()/transfer()).
2. A proposal for transaction is notified to lis_B (ReceptionListener#offered()).
3. 'B' agrees to the proposal for transaction (Session#agreed()).
4. The agreement to the proposal is notified to lis_A (ReceptionListener#agreed()).
5. 'A' consents to the contents of the agreement (Session#confirm()).
6. Completion of the transaction is notified to both of lis_A and lis_B (ReceptionListener#committed()).

The above description refers to the case where a transaction has been carried out normally (i.e. faultless cases). However, the transaction may be suspended halfway for various reasons, for example, due to a fault in the communication network, or because the partner of the transaction ignores a notification. In such a case, use of recover(), provided by the Session interface allows the user to cancel the transaction (original status prior to the transaction is recovered) or to forcibly complete the transaction. Which of the cancellation or completion will be performed depends on the progress status of the transaction.

The recovery processing by recover() is performed by asynchronous invocation. The result of the recovery processing is notified by invocation of the listener. If the transaction is cancelled, aborted() of the registered listener is invoked, and if the transaction is complete, committed() is invoked (ReceptionListener #aborted()/committed()). In either case, the fairness of transaction is assured. That is, the transaction is either completed or put back to the original status prior to the start; neither party loses its own e-rights/e-money, without receiving the e-right/e-money from the other party by reason of the transaction.

Recovery processing may involve communication with an arbitration server. Whether recovery can be achieved without the need of communicating with an arbitration server or not can be checked using isCancelable() provided by the Session interface. If its return value is true, recovery can be achieved without having to communicate with an arbitration server. In this case, the recovery processing always results in cancellation.

If communication with an arbitration server is required for recovery (if the return value of isCancelable() is false), the recover() method transmits an arbitration request to an arbitration server requesting recovery of it, and acquires the arbitration result (as to whether complete or cancelled) decided by the arbitration server. The recovery result follows the arbitration result. The arbitration server used here is one designated by Agent#setArbiter(). Since the communication with an arbitration server and recovery processing take place automatically inside recover(), APs need not take into account a communication procedure with the arbitration server.

A sample use of this API is described below:

// 1. Initialization

// 'ParticipantRepository' compatible with 'Internet Draft' (draft-ietf-trade-voucher-vtsapi-06) is created.

// 2. Authentication on the IC card

// An entry point for operation of e-rights/e-money is acquired. Here, the eTRON ID of the IC card will be taken for 'cardID,' and the instance of the class that implements AuthHandler interface will be taken for 'myAuthHandler.'

// 3. Acquisition of a folder

// Acquisition of a folder. The name of the folder will be taken for 'folderName.'

// 4. Creation of e-right/e-money

One transferable e-right/e-money having 'contents' for its contents is created.

// 5. Acquisition of e-right/e-money table inside the folder

// Since Folder inherits Set, the e-rights/e-money it possesses can be handled using Collection API provided by Java.

// All the issuers and number of the e-rights/e-money stored in the folder are output.

// 6. Transfer (Sending of e-right/e-money)

// The transferee is identified. Here, the eTRON ID of the partner will be taken for 'partnerID.'

/// e-Right/e-money 'v1' to be delivered to the partner through the transfer is identified. Here, it only is the e-right/e-money stored in the folder (in practice, the user may select one from the e-right/e-money table.)

// The listeners to be invoked when the transfer or exchange is suspended or ends are registered. Here, the instance of the class that implements the ReceptionListener interface will be taken for 'listener.'

// Execution of transfer. Three of v1 stored are sent to the partner. For example, if there are ten of v1 stored, seven of v1 will remain. Return value 's' is the ID designed to identify the transfer session. Once the transfer is complete, 'listener.committed()' will be invoked using 's' as the argument, or if the transfer is cancelled, 'listener.aborted(s)' will be invoked.

If the transfer becomes unable to continue in the middle for any reason, 'listener.suspended()' will be invoked.

// 7. Buy-sell (Exchange of e-rights/e-money)

// Execution of exchange. Three of v1 stored are fairly exchanged for v2 owned by the partner. Here, 'v2condition' is a condition that is specified for e-right/e-money v2 to be received from the partner. If the exchange is completed or cancelled, 'listener.committee()' or 'listener.aborted()' will respectively be invoked, as in the case of the transfer.

// 8. Log-out from the IC card (Transition to unauthenticated mode)

// To transit to the unauthenticated mode, it is not necessary to specify AuthHandler.

org.t_engine.tenet.vts

Interface Agent

All Superinterfaces:

Participant

public interface **Agent**
extends Participant

This interface is designed to provide an entry point for operating a TENeT-enabled IC card.
It is acquired by invoking the Participant#getAgent() method of Participant that represents an IC card. Agent has different authority modes, but can operate the IC card only in the owner mode.
To conduct an exchange or transfer on the e-rights/e-money stored on an IC card, it is essential to have an arbitration server registered. Moreover, the result of an exchange or transfer conducted on the e-rights/e-money stored on an IC card is notified, as needed, by the method of the ReceptionListener interface registered through this interface. Therefore, to exchange or transfer e-rights/e-money, it is essential to have ReceptionListener registered by the SetListener() method.

See Also::

Participant, Folder, ReceptionListener, AuthHandler

Methods

createFolder

```
public Folder createFolder(java.lang.String name,
                           int acl)
                    throws VTSInternalException,
                           VTSAccessViolationException,
                           VTSMemoryOverflowException,
                           VTSSmartcardNotFoundException,
                           VTSParameterException
```

Creates a folder with name 'name' and access right 'acl' in the smartcard. For the specified value of access right 'acl' of the folder to be created, use the value provided by the FolderAcl interface. When multiple access rights are to be assigned, the values of the constant field of the FolderAcl interface can be specified coupling them by logical OR. (For example, when the e-right/e-money creation authority and e-right/e-value transfer authority are to be assigned, createFolder(name, FolderAcl.CreateAccess | Folder Act1.TransferAccess) should be specified.)

Parameters:

`name` - Name of the folder to be created
`acl` - Access right of the folder to be created

Returns:

Folder created

Exceptions

`VTSInternalException` - An internal error occurred.
`VTSInternalException` - Agent has not obtained the necessary access right for processing.
`VTSMemoryOverflowException` - The IC card does not have enough free space.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSParameterException` - 'null' was set for the name of the folder to be created, or there already exists another folder of the same name, or access right 'acl' of the folder to be created is not correct.

deleteFolder

```
public void deleteFolder(java.lang.String name,
                         boolean mode)
                  throws VTSInternalException,
                         VTSAccessViolationException,
                         VTSObjectNotFoundException,
                         VTSSmartcardNotFoundException,
                         VTSParameterException
```

Deletes a folder with name 'name' from the smartcard, according to the specification of deletion mode 'mode.' If deletion mode 'mode' is true, it deletes the folder even if the specified folder contains e-rights/e-money. If the specified folder contains e-rights/e-money when deletion mode 'mode' is false, it throws VTSAccessViolationException.
If the folder contains the e-rights/e-money currently being exchanged, regardless of deletion mode 'mode,' it throws VTSAccessViolationException.

Parameters:

`name` - Name of the folder to be deleted
`mode` - Deletion mode

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing, or the folder contains e-rights/e-money when deletion mode 'mode' is false, or the folder contains the e-rights/e-money currently being exchanged.
`VTSObjectNotFoundException` - The folder to be deleted was not found.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSParameterException` - 'null' was set for name 'name' of the folder to be deleted.

---

getFolders

```
public org.t_engine.tenet.util.Map getFolders()
                         boolean mode)
                  throws VTSInternalException,
                         VTSAccessViolationException,
                         VTSSmartcardNotFoundException,
```

Returns a table of folders stored on the IC card as Map of the folder (Folder) using the folder name (String) as the key.

Returns:

Map of the folder (Folder) using the folder name (String) as the key .

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)

---

getSessions

```
public org.t_engine.tenet.util.Map getSessions()
                  throws VTSInternalException,
                         VTSAccessViolationException,
                         VTSSmartcardNotFoundException,
                         VTSUnsupportedMessageException
```

Returns a table of session IDs (String) of the sessions stored on the IC card and currently being exchanged or transferred, as Map of the session (Session) using the session ID (String) as the key.

Returns:

Map of the session (Session) using the session ID (String) as the key.

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSUnsupportedMessageException` - The IC card does not support this processing.

---

getListener

```
public ReceptionListener getListener()
```

Returns ReceptionListener which was registered by the setListener() method. If it is not registered, 'null' is returned.

Returns:

ReceptionListener registered

---

setListener

```
public void setListener(ReceptionListener listener)
```

Registers listeners 'listeners' for notifying the processing results of exchanges or transfers, or confirmation information. The registered listeners are invoked when an exchange/transfer request is received, when an exchange/transfer response is received, when an exchange/transfer completion notification is received, when an exchange/transfer suspension notification is received, or when an exchange/transfer error detection notification is received. Therefore, to conduct an exchange or transfer of e-rights/e-money, it is essential to have listeners registered by this method. If 'null' is specified, the registration of listeners is cleared.

Parameters:

`listener` - Listener to notify the processing result of an exchange or transfer and confirmation information.

---

getAuthMode

```
public int getAuthMode()
```

Returns the current authority mode. For the authority mode to be returned, the value provided by the AuthHandler interface is used.

Returns:

Current authority mode

---

changeAuthMode

```
public boolean changeAuthMode(AuthHandler verifier,
                              int mode)
                       throws VTSInternalException,
                              VTSAccessViolationException,
                              VTSSmartcardNotFoundException,
                              VTSParameterException
```

Changes the authority mode of Agent to the status of authority mode 'mode'. For the authority mode to be specified, the value provided by the AuthHandler interface is used. If authority mode 'mode' to be changed is the owner mode, the AuthHandler#authentication() method of the specified 'verifier' will be called back to perform authentication operation. If authority mode 'mode' to be changed is the unauthenticated mode, 'null' can be specified as ‘verifier’.
It is recommended to end the access after changing the status to the unauthenticated mode by this method.

Parameters:

`verifier` - An instance of a class implementing the AuthHandler interface
`mode` - Authority mode to be changed

Returns:
True is returned if successful.

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Authentication failed.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSParameterException` - 'null' was set when authority mode 'mode' was the owner mode, or when authority mode 'mode' was not correct.

---

getHolder

```
public Participant getHolder()
```

Returns Participant representing an IC card.

Returns:
Participant representing an IC card

---

getArbiter

```
public void setArbiter(Participant arbiter)
                throws VTSParameterException
```

Returns the arbitration server currently registered at Agent. If no registered server is found, 'null' will be returned.

Returns:
Arbitration server currently registered at Agent.

---

setArbiter

```
public void setArbiter(Participant arbiter)
                throws VTSParameterException
```

Registers the arbitration server to be used in an exchange or transfer.

Parameters:
`arbiter` - Arbitration server

Exceptions:
`VTSParameterException` - 'null' was set for arbitration server 'arbiter'.

org.t_engine.tenet.vts

Interface AuthHandler

public interface **AuthHandler**

This interface abstracts the function to perform authentication processing for changing the authority mode.
This interface also defines values to specify the authority mode.

See Also:

Agent

Fields

None

```
public static final int None
```

Unauthenticated mode

Owner

```
public static final int Owner
```

Owner mode

Methods

getAuthentication

```
public byte[] getAuthentication(byte[] challenge)
                         throws VTSParameterException,
                                VTSInternalException
```

This method is invoked to generate authentication information. The algorithm for creation of authentication information needs to be implemented in the class that implements this interface.

Parameters:

`challenge` - Challenge information to generate authentication information

Returns:

Authentication information

Exceptions:

`VTSParameterException` - Challenge information of authentication 'challenge' is 'null', or the size is not correct.

`VTSInternalException` - An internal error occurred during the creation of authentication information.

org.t_engine.tenet.vts

Interface FileAcl

public interface **FileAcl**

This interface is designed to manage the access right to e-rights/e-money.
The access right is applied to other parties than the issuer who created the e-rights/e-money. Therefore, the issuer can perform all the operations, irrespective of whether it possesses the access right or not. This interface also defines the specified value of the access right.

See Also:

StoredVoucher

Fields

CopyAccess

```
public static final int CopyAccess
```

Copy right

TransferAccess

```
public static final int TransferAccess
```

Transfer right

Methods

isDuplicatable

```
public boolean isDuplicatable()
```

Returns true when e-rights/e-money are duplicatable.

Returns:

True, if e-rights/e-money are duplicatable.

isTransferable

```
public boolean isTransferable()
```

Returns true if e-right/e-money are transferable.

Returns:

True, if e-rights/e-money are transferrable.

org.t_engine.tenet.vts

Interface Folder

public interface **Folder**
extends org.t_engine.tenet.util.Set

This interface is designed to provide an entry point for operating the folders stored on an IC card.
It is acquired invoking the Agent#getFolders() method. Folder is an aggregation of StoredVoucher, and it in itself succeeds to the Set interface.

See Also::
Agent, FolderAcl

Methods

getName

```
public java.lang.String getName()
```

The name of this folder is returned.

Returns:
Name of this folder

createVoucher

```
public StoredVoucher createVoucher(byte[] promise,
                                   int num,
                                   int acl)
                            throws VTSInternalException,
                                   VTSAccessViolationException,
                                   VTSObjectNotFoundException,
                                   VTSMemoryOverflowException,
                                   VTSMaximumNumberException,
                                   VTSSmartcardNotFoundException,
                                   VTSParameterException
```

Creates e-rights/e-money inside an IC card by defining, as the issuer, the eTRONID of the IC card having contents 'promise' of the e-rights/e-money to be created, number 'num' of the e-rights/e-money to be created, access right 'acl' to the e-rights/e-money to be created, and Folder. For the specified value of access right 'acl' to the e-rights/e-money to be created, use the value provided by the FileAcl interface. If multiple access rights are to be assigned, the values of the constant field of the FileAcl interface can be specified coupling them by logical OR. (For example, if the copy authority and transfer authority are to be assigned, createVoucher(promise, num, FileAcl.CopyAccess | FileAcl.TransferAccess) can be specified.)

Parameters:
`promise` - Contents of the e-rights/e-money to be created.
`num` - Number of the e-rights/e-money to be created.
`acl` - Access right of the e-rights/e-money to be created.

Returns:
e-Eights/e-money stored on the IC card.

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSObjectNotFoundException` - There is no folder to be operated.
`VTSMemoryOverflowException` -The IC card does not have enough free space, or the size of the e-rights/e-money exceeds the maximum storable value.

VTSMaximumNumberException - The number of e-rights/e-money exceeds the maximum number.
VTSSmartcardNotFoundException - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
VTSParameterException - Content 'promise' of the e-rights/e-money to be created is 'null,' or number 'num' of the e-rights/e-money to be created is 0 or less, or access right 'acl' of the e-rights/e-money to be created is not correct.

---

getIdentifier

`public java.lang.String` **`getIdentifier`**`()`

Returns the ID of this folder.

Returns:
ID of this folder

---

getFolderAcl

`public` FolderAcl **`getFolderAcl`**`()`

Returns the access right to this folder.

Returns:
Access right to this folder

---

iterator

`public org.t_engine.tenet.util.Iterator` **`iterator`**`()`

Returns a collection of the e-rights/e-money possessed by this folder. If acquisition of the e-rights/e-money fails because there is no response to the IC card, the necessary access right does not exist, or for other reasons, 'null' is returned. If there exists no e-right/e-money in this folder, an empty collection is returned.

Returns:
Collection of Iterator e-rights/e-money, or 'null' if acquisition of e-rights/e-money fails.

---

iterator

`public org.t_engine.tenet.util.Iterator` **`iterator`**`(VoucherCondition condition)`

Returns a collection of e-rights/e-money that meet e-rights/e-money extraction condition 'condition.' If the acquisition of e-rights/e-money fails because there is no response to the IC card, the necessary access right does not exist or for any other reason, 'null' is returned. If there is no e-right/e-money that meets e-right/e-money extraction condition 'condition,' an empty collection is returned.

Parameters:
`condition` - e-Right/e-money extraction condition

Returns:
Collection of Iterator e-rights/e-money, or 'null' if acquisition of e-rights/e-money fails.

---

org.t_engine.tenet.vts

Interface FolderAcl

public interface **FolderAcl**

This interface is designed to hold the access rights to folders.
The access right applies to other parties than the owner. Therefore, the owner can perform all the operations, irrespective of whether it possesses the access right or not. This interface also defines the specified value of the access right.

See Also:

Folder

Fields

ReadAccess

```
public static final int ReadAccess
```

Authority to read folders .

CreateAccess

```
public static final int CreateAccess
```

Authority to create e-rights/e-money.

TransferAccess

```
public static final int TransferAccess
```

Authority to transfer e-rights/e-money.

Methods

isReadable

```
public boolean isReadable()
```

Returns true if the folder is readable.

Returns:

True, if the folder is readable.

isCreateable

```
public boolean isCreateable()
```

Returns true if e-rights/e-money can be created inside the folder.

Returns:

True, if e-rights/e-money are creatable inside the folder.

isTransferable

```
public boolean isTransferable()
```

Returns true if e-rights/e-money in the folder are transferable.

Returns:

True, if e-rights/e-money in the folder are transferable.

org.t_engine.tenet.vts

Interface Participant

All Subinterfaces:

Agent

public interface **Participant**

This interface represents an entity with eTRONID.
It represents the entities that are assigned eTRONID, such as TENeT IC cards, arbitration servers, and applications that serve as partners in the transactions of e-rights/e-money. The interface is acquired invoking the ParticipantRepository#lookup () method.

See Also:

ParticipantRepository, Agent, AuthHandler

Methods

getIdentifier

```
public java.lang.String getIdentifier()
```

Returns eTRONID of this Participant.

Returns:

eTRONID OF this Participant

getAgent

```
public Agent getAgent(AuthHandler verifier,
                      int mode)
               throws VTSInternalException,
                      VTSAccessViolationException,
                      VTSSmartcardNotFoundException,
                      VTSParameterException
```

Creates/returns Agent for accessing Participant. It can be executed only on Participant that represents an IC card (with eTRONID of an IC card). Specify the value provided by the AuthHandler interface for the value to be specified in authority mode 'mode.' If authority mode 'mode' is the owner mode, the AuthHandler#authentication() method of interface equipped with authentication algorithm 'verifier' will be called back to conduct authentication operation. If authority mode 'mode' is the unauthenticated mode, interface equipped with authentication algorithm 'verifier' is not used, and therefore, 'null' can be specified. By the way, for interface equipped with authentication algorithm 'verifier,' the equipping needs to be defined on the AP side. To terminate an IC card access when Agent has been acquired, with the owner mode specified for authority mode 'mode,' it is recommended to end the access after changing the status to the unauthenticated mode by the Agent#changeAuthMode() method.

Parameters:

`verifier` - Interface equipped with authentication algorithm
`mode` - Authority mode

Returns:

Agent designed to access Participant.

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Authentication failed.
`VTSSmartcardNotFoundException` -Participant is not the one that represents an IC card, or a TENeT IC card capable of dispensing eTRONID is not found.

`VTSParameterException` -'null' was set for interface equipped with authentication algorithm 'verifier' when authority mode 'mode' was the owner mode, or authority mode 'mode' is not correct.

org.t_engine.tenet.vts

Interface ParticipantRepository

public interface **ParticipantRepository**

This interface is designed to create/manage Participant.
To be able to use this API, it is necessary to acquire the instance of the class that implements this interface.

See Also:
`Participant`

Methods

lookup

```
public Participant lookup(java.lang.String eTronID)
                   throws VTSParameterException
```

Creates and returns Participant that represents the eTRONID of Participant to be created.

Parameters:
`ETronID` - eTRONID of Participant to be created.

Returns:
Participant representing the eTRONID of Participant to be created .

Exceptions:
`VTSParameterException` - 'null' was set for the eTRONID of Participant to be created.

org.t_engine.tenet.vts

Interface ReceptionListener

public interface **ReceptionListener**

This interface is designed to receive the notification of results of an exchange or transfer conducted on the e-rights/e-money stored on an IC card.
To conduct an exchange or transfer on e-rights/e-money, it is essential to prepare a class that implements this interface and register its instances at Agents.

See Also:

`Agent`, `Session`

Methods

offered

```
public void offered(Session offer,
                    byte[] receivingVoucherCondition)
```

Notifies that an exchange request was received. Check the session offer for the exchange request and exchange condition for e-rights/e-money 'receiving VoucherCondition,' required by the partner, and agree to the presented request by using the Session#agree() method. If the presented request is to be rejected, use the Session#reject() method. In this case, the partner is notified by the suspended() method.

Parameters:

`offer` - Session for the exchange request
`receivingVoucherCondition` - Exchange conditions for e-rights/e-money, required by the partner.

agreed

```
public void agreed(Session session)
```

Notifies that a response to the transfer request was received. Check session 'session' for the transfer response and confirm the transfer by using the Session#confirm() method. To reject the transfer response, use the Session#reject() method. In this case, the partner is notified by the suspended() method.

Parameters:

`session` – Session for transfer response

agreed

```
public void agreed(Session session,
                   Voucher receivingVoucher)
```

Notifies that a response to the exchange request was received. Check session 'session' for the exchange response and confirm e-rights/e-money 'receiving Voucher,' to be exchanged, designated by the partner, and confirm the exchange by using the Session#confirm() method. To reject the exchange response, use the Session#reject() method. In this case, the partner is notified by the suspended() method.

Parameters:

`session` - Session for exchange response
`receivingVoucher` - e-Rights/e-money to be exchanged, designated from the partner side.

committed

```
public void committed(Session session)
```

Notifies that an exchange or transfer is complete with respect to session 'session' for the exchange or transfer. Reception of this notification enables the exchanged e-rights/e-money to be used.

Parameters:
`session` - Session for the exchange or transfer

---

aborted

```
public void aborted(Session session,
                    int causeCode)
```

Notifies that an exchange or transfer was cancelled, with respect to session 'session' for the exchange or transfer. If this method is invoked, it indicates that the exchange or transfer represented by session 'session' has been cancelled, or it has been suspended by an arbitration server. The reason for the cancellation shall be able to be judged by cancellation cause code 'causeCode' specified with an argument. The value of the constant field defined by the Session interface is specified for the cancellation cause code.

Parameters:
`session` - Session for the exchange or transfer
`causeCode` - Cancellation cause code

---

suspended

```
public void suspended(Session session,
                      int causeCode)
```

Notifies of the detection of an error in an exchange or transfer. It indicates that an error has been detected for session 'session' for an exchange or transfer. The reason for error detection shall be able to be judged by the error detection cause code 'causeCode' specified with an argument. The value of the constant field defined by the Session interface is specified for the error detection cause code. If the error detection cause code indicates "rejection," it is possible to acquire the reason for the rejection by invoking the Session#getRejectReason() method. If this method is invoked, it is necessary to invoke the Session#recover() method to recover the exchange.

Parameters:
`session` - Session for the exchange or transfer
`causeCode` - Error detection cause code

org.t_engine.tenet.vts

Interface Session

public interface **Session**

This interface is designed to hold the information of an exchange or transfer. The interface also provides operations (agreement, confirmation, recovery, etc.) for an exchange or transfer.
It is created when an exchange or transfer is conducted on the e-rights/e-money stored on an IC card. This interface holds a unique ID up until the exchange or transfer is complete, irrespective of whether the exchange or transfer processing is normal or abnormal.
To recover an exchange or transfer, the recover() method needs to be invoked, or for the side requested (proposed) of an exchange or transfer to agree to it, the agree() method needs to be invoked, and for the side that requested (proposed) an exchange or transfer to confirm it, the confirm() method needs to be invoked. Moreover, for both of the requesting side and requested (proposed) side to reject, the reject() method has to be invoked.

See Also:

Voucher, Participant

Fields

None

`public static final int` **`None`**

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "No suspension or error detection has occurred."

AbortedByRecovery

`public static final int` **`AbortedByRecovery`**

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "The exchange has been cancelled by the arbitration server."

ExchangeRejected

`public static final int` **`ExchangeRejected`**

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "Rejected by the side that requested (proposed) the exchange or transfer."

AccessViolation

`public static final int` **`AccessViolation`**

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "Agent has not obtained the necessary access right for processing."

ObjectNotFound

`public static final int` **`ObjectNotFound`**

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "There does not exist the folder or e-right/e-money."

IllegalParameters

`public static final int` **`IllegalParameters`**

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is:

"The exchange cannot be confirmed with the specified e-right/e-money."

MaximumNumberExceeded

```
public static final int MaximumNumberExceeded
```

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "The number of e-rights/e-money is insufficient."

MemoryOverflow

```
public static final int MemoryOverflow
```

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "There is not enough space to store exchange information on the IC card."

IncompatibleStatus

```
public static final int IncompatibleStatus
```

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "Unable to continue because there is no exchange processing, or the status is illegal."

InternalError

```
public static final int InternalError
```

It indicates that the cause of occurrence in a notification of suspension or notification of error detection is: "An internal error occurred."

Methods

isOriginator

```
public boolean isOriginator()
                    throws VTSInternalException,
                           VTSAccessViolationException,
                           VTSObjectNotFoundException,
                           VTSSmartcardNotFoundException,
                           VTSUnsupportedMessageException
```

Returns true if this session corresponds to the side that requested (proposed) the exchange or transfer.

Returns:

True, if this session corresponds to the side that requested (proposed) the exchange or transfer.

Exceptions:

`VTSInternalException` - An internal error occurred.

`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.

`VTSObjectNotFoundException` - The exchange information to be operated no longer exists on the IC card.

`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)

`VTSUnsupportedMessageException` - The IC card does not support this processing.

isRecipient

```
public boolean isRecipient()
                   throws VTSInternalException,
                          VTSAccessViolationException,
                          VTSObjectNotFoundException,
```

```
                                VTSSmartcardNotFoundException,
                                VTSUnsupportedMessageException
```

Returns true if this session corresponds to the side that was requested (proposed) of the exchange or transfer.

Returns:

True, if this session corresponds to the side requested (proposed) of the exchange or transfer.

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSObjectNotFoundException` - The exchange information to be operated no longer exists on the IC card.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSUnsupportedMessageException` - The IC card does not support this processing.

---

agree

```
public void agree(Folder incomingFolder,
                  StoredVoucher sendingVoucher)
           throws VTSInternalException,
                  VTSParameterException,
                  VTSIncompatibleStatusException
```

Agrees to the exchange or transfer represented by this session. Invoke this method to agree to an exchange or transfer notified by the ReceptionListener#offered() method, and specify the storing location for the e-rights/e-money to be delivered by the partner and the e-rights/e-money to be delivered by yourself.

Parameters:

`IncomingFolder` - Folder to store the e-rights/e-money transmitted by reason of an exchange or transfer.
`SendingVoucher` - e-Rights/e-money to be delivered by yourself.

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSParameterException` - 'null' was set for folder 'incomingFolder' storing the e-rights/e-money to be transferred.
`VTSIncompatibleStatusException` - Transfer or exchange processing could not be executed, because the internal information of the session was illegal.

confirm

```
public void confirm(Folder incomingFolder)
            throws VTSInternalException,
                   VTSParameterException,
                   VTSIncompatibleStatusException
```

Confirms the exchange or transfer represented by this session. Invoke this method to confirm the exchange or transfer notified by the ReceptionListener#agreed() method, and specify the storing location for the e-rights/e-money to be delivered by the partner. If there is no e-right/e-money to be delivered by the partner, specify 'null' for folder 'incomingFolder' that is to store the e-rights/e-money to be transmitted by reason of the exchange.

Parameters:

`IncomingFolder` - Folder to store e-rights/e-money to be transmitted by reason of the exchange.

Exceptions:

`VTSInternalException` - An internal error occurred.
`VTSParameterException` - whereas there are e-rights/e-money to be delivered by the partner, 'null' was set for folder 'incomingFolder' that is to store the e-rights/e-money to be transmitted by reason

of the exchange.
`VTSIncompatibleStatusException` - Transfer or exchange processing could not be executed, because the internal information of the session was illegal.

---

reject

```
public void reject(java.lang.String rejectReason)
            throws VTSInternalException
```

Rejects the exchange or transfer represented by this session. Invoke the method to reject the response to an exchange or transfer notified by the ReceptionListener#offered() or agreed() method, and specify the reason for the rejection. It is notified by ReceptionListener#suspended() method to the partner of the exchange or transfer. The notified partner can acquire rejection reason 'rejectReason' by the Session#getRejectReason() method. Specify 'null' if there is no reason for the rejection.

Parameters:
`rejectReason` - Reason for rejection

Exceptions:
`VTSInternalException` - An internal error occurred.

---

recover

```
public void recover()
             throws VTSInternalException
```

Cancels the exchange or transfer represented by this session, or requests recovery of an arbitration server. Whether cancellation is done or recovery request is made of an arbitration server depends on the status of the session held by the IC card. To receive the completion notification for a recovery request, suspension notification or error detection notification, it is essential to have ReceptionListener registered.

Exceptions:
`VTSInternalException` - An internal error occurred.

---

getIdentifier

```
public java.lang.String getIdentifier()
```

Returns the ID of this session.

Returns:
ID of this session.

---

getPartner

```
public Participant getPartner()
```

Returns the partner of the exchange or transfer represented by this session. If this session corresponds to the side that requested (proposed) the exchange or transfer, return Participant of the side that was requested (proposed) of the exchange or transfer. If this session corresponds to the side that was requested (proposed) of the exchange or transfer, return Participant of the side that requested (proposed) for the exchange or transfer. If no such Participant is found, return 'null' (for example, in the case where the exchange was suspended and it is no longer managed by the IC card).

Returns:
Partner of the intended exchange or transfer

---

getSendingVoucher

```
public Voucher getSendingVoucher()
```

Returns the e-right/e-money to be transmitted in an exchange or transfer represented by this session. If no such e-right/e-money exists, return 'null' (in a transfer, for example).

Returns:
e-Right/e-money to be transmitted in an exchange or transfer represented by this session.

---

getReceivingVoucher

```
public Voucher getReceivingVoucher()
```

Returns the e-right/e-money to be received in an exchange or transfer represented by this session. If no such e-right/e-money exists, return 'null' (in a transfer, for example).

Returns:
e-Right/e-money to be received in an exchange or transfer represented by this session.

---

isCancelable

```
public boolean isCancelable()
                    throws VTSInternalException,
                           VTSAccessViolationException,
                           VTSSmartcardNotFoundException,
                           VTSUnsupportedMessageException
```

Returns true if this session is cancelable.

Returns:
True, if this session is cancelable.

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSUnsupportedMessageException` - The IC card does not support this processing.

---

isRecoverable

```
public boolean isRecoverable()
                    throws VTSInternalException,
                           VTSAccessViolationException,
                           VTSSmartcardNotFoundException,
                           VTSUnsupportedMessageException
```

Returns true if this session is recoverable.

Returns:
True, if this session is recoverable.

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC card.)
`VTSUnsupportedMessageException` - The IC card does not support this processing.

---

getRejectReason

```
public java.lang.String getRejectReason()
```

Returns the reason for rejection in the case where this session received an error detection notification (rejection). The reason for rejection is the one which was specified when the exchange or transfer was rejected by the partner. If there is no reason for rejection, a character string of 0 byte is returned.

Returns:
Reason for rejection when this session received an error detection notification (rejection).

org.t_engine.tenet.vts

Interface StoredVoucher

All Superinterfaces:
Voucher

public interface **StoredVoucher**
extends Voucher

This interface is designed to provide an entry point for operating the e-rights/e-money stored on an IC card.
This interface represents the e-rights/e-money stored on the IC card. It is acquired extracting it by the Folder#createVoucher method and Folder#iterator() method.

See Also:
Folder, Voucher

Methods

transfer

```
public Session transfer(Participant destination)
                throws VTSInternalException,
                       VTSParameterException
```

Transfers this e-right/e-money to transfer partner 'destination.' To receive a transfer response to the transfer request, completion notification, or error detection notification, it is essential to have ReceptionListener registered. Moreover, if no response comes back from the transfer destination, the transfer request can be recovered invoking the Session#recover() method of the session, which is a return value.

Parameters:
`destination` - Transfer destination

Returns:
Session for the transfer request

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSParameterException` - 'null' was set for transfer destination 'destination'.

exchange

```
public Session exchange(Participant destination,
                        byte[] receivingVoucherCondition)
                throws VTSInternalException,
                       VTSParameterException
```

Exchanges this e-right/e-money for the one corresponding to exchange condition 'receivingVoucher' with exchange partner 'destination.' To receive an exchange response to the exchange request, completion notification, or error detection notification, it is essential to have ReceptionListener registered. Moreover, if no response comes back from the exchange partner side, the exchange request can be recovered invoking the Session#recover() method of the session, which is a return value of this method. If there is no exchange condition, specify 'null' for exchange condition 'receivingVoucherCondition.'

Parameters:
`destination` - Exchange partner
`receivingVoucherCondition` - Exchange condition

Returns:

Session for the exchange request.

Exceptions:
`VTSInternalException` - An internal error occurred, or there is no arbiter set up at Agent.
`VTSParameterException` - 'null' was set for exchange partner 'destination.'

---

move

```
public StoredVoucher move(Folder folder)
                  throws VTSInternalException,
                         VTSAccessViolationException,
                         VTSObjectNotFoundException,
                         VTSMemoryOverflowException,
                         VTSMaximumNumberException,
                         VTSParameterException,
                         VTSSmartcardNotFoundException
```

Moves this e-right/e-money to folder 'folder' located at the movement destination.

Parameters:
`folder` - Folder at the movement destination.

Returns:
e-Right/e-money after the movement.

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSAccessViolationException` - Agent has not obtained the necessary access right for processing.
`VTSObjectNotFoundException` - There is no e-right/e-money or folder to be operated.
`VTSMemoryOverflowException` - The IC card does not have enough free space, or the size of the e-right/e-money exceeds the maximum storable value.
`VTSMaximumNumberException` - The number of e-rights/e-money at the movement origin is insufficient or the number of e-rights/e-money at the movement destination exceeds the maximum number.
`VTSParameterException` - 'null' was set for folder 'folder' at the movement destination, or the same folder as that of the movement origin was set at the movement destination.
`VTSSmartcardNotFoundException` - There is no response to the IC card access. (Generally, this occurs when there is no IC cared.)

---

copy

```
public StoredVoucher copy(Folder folder)
                  throws VTSInternalException,
                         VTSAccessViolationException,
                         VTSObjectNotFoundException,
                         VTSMemoryOverflowException,
                         VTSMaximumNumberException,
                         VTSParameterException,
                         VTSSmartcardNotFoundException
```

Copies this e-right/e-money to folder 'folder' located at the copy destination.

Parameters:
`folder` - Folder at the copy destination

Returns:
e-Right/e-money after copying

Exceptions:
`VTSInternalException` - An internal error occurred.

VTSAccessViolationException - Agent has not obtained the necessary access right for processing, or has no authority to copy this e-right/e-money.
VTSObjectNotFoundException - There is no e-right/e-money or folder to be operated.
VTSMemoryOverflowException - The IC card does not have enough free space, or the size of the e-rights/e-money exceeds the maximum storable value.
VTSMaximumNumberException - The number of e-rights/e-money at the movement origin is insufficient, or the number of e-rights/e-money at the movement destination exceeds the maximum number.
VTSParameterException - 'null' was set for folder 'folder' at the movement destination, or the same folder as that of the movement origin was set at the movement destination.
VTSSmartcardNotFoundException - There is no response to the IC card access. (Generally, this occurs when there is no IC cared.)

---

remove

```
public void remove(int num)
            throws VTSInternalException,
                   VTSAccessViolationException,
                   VTSObjectNotFoundException,
                   VTSMaximumNumberException,
                   VTSParameterException,
                   VTSSmartcardNotFoundException
```

Removes this e-right/e-money.

Parameters:
num - number of e-rights/e-money to be deleted.

Exceptions:
VTSInternalException - An internal error occurred.
VTSAccessViolationException - Agent has not obtained the necessary access right for processing.
VTSObjectNotFoundException - There is no e-right/e-money or folder to be operated.
VTSMaximumNumberException - A larger value than the existing number was set for number to be deleted 'num.'
VTSParameterException - A value smaller than 0 was set for number to be deleted 'num.'
VTSSmartcardNotFoundException - There is no response to the IC card access. (Generally, this occurs when there is no IC cared.)

---

getFolder

```
public Folder getFolder()
```

Returns the folder storing this e-right/e-money.

Returns:
Folder storing this e-right/e-money

---

getIdentifier

```
public java.lang.String getIdentifier()
```

Returns the identifier of this e-right/e-money.

Returns:
Identifier of this e-right/e-money

---

select

```
public StoredVoucher select(int num)
                    throws VTSParameterException,
                           VTSInternalException
```

Returns the e-rights/e-money equivalent to selected number 'num.' If selected number 'num' is smaller than 0 or larger than the number currently possessed, throw VTSParameterException.

Parameters:
`num` - Selected number.

Returns:
e-Rights/e-money equivalent to selected number 'num.'

Exceptions:
`VTSInternalException` - An internal error occurred.
`VTSParameterException` - A value smaller than 0 or larger than the number currently possessed was set for selected number 'num.'

org.t_engine.tenet.vts

Interface Voucher

All Subinterfaces:
StoredVoucher

public interface **Voucher**

This interface represents e-rights/e-money.
This interface holds the elements composing e-rights/e-money.

See Also:
Participant, StoredVoucher

Methods

getIssuer

`public Participant getIssuer()`
Returns the issuer of this e-right/e-money.

Returns:
The issuer of this e-right/e-money.

getPromise

`public byte[] getPromise()`
Returns the content of this e-right/e-money.

Returns:
Content of this e-right/e-money

getCount

`public int getCount()`
Returns the number of these e-rights/e-money.

Returns:
Number of these e-rights/e-money

getFileAcl

`public FileAcl getFileAcl()`
Returns the access right to this e-right/e-money.

Returns:
Access right to this e-right/e-money

org.t_engine.tenet.vts

Interface VoucherCondition

public interface **VoucherCondition**

This interface provides a search interface designed to search for e-rights/e-money.
By specifying a class that implements this interface in the parameters of the Folder#iterator() method, it is possible to search (filtering) for e-rights/e-money.

See Also:

Voucher, Participant

Methods

matches

`public boolean` **`matches`**`(Voucher object)`

Returns true if e-right/e-money to be compared 'object' meets this exchange condition. Normally, the e-right/e-money stored on the IC card (StoreVoucher) is specified for e-right/e-money to be compared 'object.'

Parameters:

`Object` - e-Right/e-money to be compared

Returns:

True, if e-right/e-money to be compared 'object' meets this exchange condition.

org.t_engine.tenet.vts

Class VTSAccessViolationException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSAccessViolationException
```

public class **VTSAccessViolationException**
extends VTSException

This is thrown if the IC card suspends the processing because of the absence of the specified e-right/e-money or the access right to the folder. It is thrown also when authentication fails.

Constructors

VTSAccessViolationException

`public VTSAccessViolationException(java.lang.String string)`

Builds up VTSAccessViolationException with detailed message 'string.'

Parameters:

`string` - Detailed message

org.t_engine.tenet.vts

Class VTSException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
+-org.t_engine.tenet.vts.VTSException
```

Direct Known Subclasses:

VTSAccessViolationException, VTSIncompatibleStatusException, VTSInternalException, VTSMaximumNumberException, VTSMemoryOverflowException, VTSMessageSizeOverflowException, VTSObjectNotFoundException, VTSParameterException, VTSSmartcardNotFoundException, VTSUnsupportedMessageException

---

public class **VTSException**
extends java.lang.Exception

This is the base class of an exception that is thrown if the exception occurs at vts.

Constructors

VTSException

`public` **`VTSException`**`(java.lang.String string)`

Builds up VTSException with detailed message 'string.'

Parameters:

`string` - Detailed message

org.t_engine.tenet.vts

Class VTSIncompatibleStatusException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSIncompatibleStatusException
```

public class **VTSIncompatibleStatusException**
extends VTSException

This is thrown if the specified exchange or transfer is not found or if the status of the exchange or transfer is illegal.

Constructors

VTSIncompatibleStatusException

`public` **`VTSIncompatibleStatusException`**`(java.lang.String string)`

Builds up VTSIncompatibleStatusException with detailed message 'string.'

Parameters:

`string` - Detailed message

org.t_engine.tenet.vts

Class VTSInternalException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSInternalException
```

public class **VTSInternalException**
extends VTSException

This is thrown if an internal error occurs at vts.
For example, it is thrown in the case where the IC card notifies of a write error.

Constructors

VTSInternalException

`public VTSInternalException(java.lang.String string)`
Builds up VTSIInternalException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSMaximumNumberException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSMaximumNumberException
```

public class **VTSMaximumNumberException**
extends VTSException

This is thrown if the IC card suspends the processing, because processing in excess of the upper-/lower-limit value of the number of e-rights/e-money was requested.
For example, it is thrown in the case where the number of e-rights/e-money at the movement destination exceeds the storable limit if e-rights/e-money are moved there.

Constructors

VTSMaximumNumberException

`public VTSMaximumNumberException(java.lang.String string)`
Builds up VTSMaximumNumberException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSMemoryOverflowException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSMemoryOverflowException
```

public class **VTSMemoryOverflowException**
extends VTSException

This is thrown if the IC card suspends the processing, because processing in excess of the upper-/lower-limit relating to the area was requested.
For example, it is thrown in the case where the size of the e-rights/e-money to be created exceeds the maximum storable value, or where no further folder can be created due to lack of the area.

Constructors

VTSMemoryOverflowException

`public` **`VTSMemoryOverflowException`**`(java.lang.String string)`
Builds up VTSMemoryOverflowException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSMessageSizeOverflowException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSMessageSizeOverflowException
```

public class **VTSMessageSizeOverflowException**
extends VTSException

This is thrown if the IC card suspends the processing, because processing in excess of the response size which the IC card can transmit or receive was requested.
For example, it is thrown in the case where the acquired result of an e-rights/e-money table exceeds the transmittable or receivable response size.

Constructors

VTSMessageSizeOverflowException

```
public VTSMessageSizeOverflowException(java.lang.String string)
```

Builds up VTSMessageSizeOverflowException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSObjectNotFoundException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSObjectNotFoundException
```

public class **VTSObjectNotFoundException**
extends VTSException

This is thrown if the IC card suspends the processing, because the right/e-money or folder to be operated was not found.

Constructors

VTSObjectNotFoundException

`public VTSObjectNotFoundException(java.lang.String string)`
Builds up VTSObjectNotFoundException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSParameterException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSParameterException
```

public class **VTSParameterException**
extends VTSException

This is thrown if the argument of the method is illegal.
For example, it is thrown in the case where 'null' was specified for a parameter for which 'null' is not permitted.

Constructors

VTSParameterException

```
public VTSParameterException(java.lang.String string)
```

Builds up VTSParameterException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSSmartcardNotFoundException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSSmartcardNotFoundException
```

public class **VTSSmartcardNotFoundException**
extends VTSException

This is thrown if time-out occurs before a response to the IC card is received.
Generally, it is thrown in the case where an IC card access is made whereas no IC card exists.

Constructors

VTSSmartcardNotFoundException

`public` **`VTSSmartcardNotFoundException`**`(java.lang.String string)`

Builds up VTSSmartcardNotFoundException with detailed message 'string.'

Parameters:
`string` - Detailed message

org.t_engine.tenet.vts

Class VTSUnsupportedMessageException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.vts.VTSException
              |
              +-org.t_engine.tenet.vts.VTSUnsupportedMessageException
```

public class **VTSUnsupportedMessageException**
extends VTSException

This is thrown if operation of an exchange-related function is conducted using an IC card that does not support the exchange-related function.

Constructors

VTSUnsupportedMessageException

`public` **`VTSUnsupportedMessageException`**`(java.lang.String string)`

Builds up VTSUnsupportedMessageException with detailed message 'string.'

Parameters:

`string` - Detailed message

Index

About Appendix org.t_engine.util.*

Both of the e-right/e-money transaction API and e^{2}TP messaging API make use of three types of interfaces, namely, org.t_engine.util.Set, org.t_engine.util.Map, and org.t_engine.util.Iterator as the interfaces for handling aggregations and associative arrays and for handling itirators.

These interfaces shall, respectively, provide the same interfaces as those of the same names that exist in java.util packages (java.util.Set, java.util.Map, and java.util.Iterator), with the following exception:

Namely, for methods in which java.util.Collection appears as an argument or return value, org.t_engine.util.Collection shall be used instead. The org.t_engine.util.Collection interface provides the same interface as java.util.Collection.

The above is a remedial measure adopted because JCF (Java Collections Framework) cannot be used in the J2ME CLDC environment.