e^{2}TP Messaging API Specification

TEF950-S002-01.00.00/en

2005/03

T-Engine Forum

e^{2}TP　Messaging API Specification

Scope

This manual defines the e^{2}TP Messaging API, which is the JAVA API specification for sending and receiving e^{2}TP (extended eTRON Transfer Protocol) messages in the TENeT standard from Java programs

Normative references

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

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

Introduction

e^{2}TP Messaging API provides interfaces to send and receive e^{2}TP messages, the communication messages used by TENeT among smartcards and application programs (AP).

This API makes it possible to send and receive e^{2}TP messages either synchronously (awaits blocks of the answerback messages following the sending of the message) or asynchronously (continues the process without waiting for the answerback message after sending a message) (see DispatchAgent#syncSend()/send()).　When a message is sent asynchronously, the answerback message to the sent messages are to be acquired through listeners registered in advance (MessageListener#received())

Any AP, IC card or server with an eTRON ID in the TENeT name space can be specified as the destination of the e^{2}TP message sent by this API.　When the destination is remote, the messages are routed automatically and are sent through an appropriate route in a network.　The AP does not need to be aware of whether the destination is local or remote.　For security reasons, however, some messages are prohibited to be sent remotely.　Refer to the TENeT message specification for the types of messages that are prohibited to be sent remotely.

In this API, the eTRON ID of an AP is generated and acquired automatically by a smartcard using the eTRON ID of the card (see SystemManager#getAgent()).　The aquired ID is stored in the DispatchAgent object and automatically used as the source ID of a message sent by the AP.　As a result, each AP don’t have to be aware of specific assignment of its eTRON ID.

Usage samples of this API are indicated here following.

```
//1.    Acquisition of an eTRON ID (acquisition of entry point for delivering and receiving messages)

String domain =
(String) SystemManager.getInstance().getDomainMap().get(IccRwName);
// Domain acquisition.    The IccRwName is the name for identifying the IC cards within the environment, such as R/W names in the PC/ SC.

DispatchAgent agent =
  SystemManager.getInstance().getAgent(domain);
// Entry point acquisition for message sending and receiving.    Here the eTRON ID is automatically issued for the AP.    Thereafter, when a message is sent, the ID issued here is automatically used as the source ID of    messages sent by the AP (The AP cannot misrepresent the source ID).

//2.    Message sending and receiving

ETronID dst = new ETronID (domain, 0);
// The eTRON ID of the IC card is port 0.

Int createFile = 0x0040;
// CreateFile message (see TENeT message specification)

Message msg = new Message (dst, createFile, fileData);
// Creation of the message to be sent.    The fileData is the data for the authorization value to be created.

Message reply = agent.syncSend(msg);
// Synchronous message sending and receiving.    It sends a message, and then waits until a message with the same Thread ID is sent to the AP address itself.    The return value is the message received.

ThreadID tid = agent.send(msg);
// Asynchronous message sending and receiving.   After it sends a msg, it immediately returns the associated Thread ID, and then returns the processing.

agent.setListener(listener);

// Register a listener for asynchronous message receiving.   When a message whose destination is this AP is received, listener.received() is invoked with the received message as the argument (the listener should be the instance of the class that implements the MessageListener interface).
```

org.t_engine.tenet.core

Class CoreErrorMessageException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.core.CoreException
              |
              +-org.t_engine.tenet.core.CoreErrorMessageException
```

public class **CoreErrorMessageException**
extends `CoreException`

It is thrown when unexpected messages arrive

Constructors

CoreErrorMessageException

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

Structures a CoreErrorMessageException that has a detailed message string.

Parameters:

`String` - Detailed message

org.t_engine.tenet.core

Class CoreException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.core.CoreException
```

Direct Known Subclasses:

`CoreErrorMessageException`, `CoreInternalException`, `CoreParameterException`, `CoreSmartcardException`

---

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

This is an underlying class of exceptions that are thrown when an exception has occurred in the core.

Constructors

CoreException

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

It structures a core exception that has a detailed message string.

Parameters:

`string` - Detailed message

org.t_engine.tenet.core

Class CoreInternalException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-org.t_engine.tenet.core.CoreException
              |
              +-org.t_engine.tenet.core.CoreInternalException
```

public class **CoreInternalException**
extends `CoreException`

It is thrown, when an internal error has been generated in the core.
For example, it is thrown when there is a failure in an internal resource file access, an illegally set file, or a failure has been occurred in an internal class generation.

Constructors

CoreInternalException

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

It structures a CoreInternalException that has a detailed message string.

Parameters:

string - detailed message

org.t_engine.tenet.core

Class CoreParameterException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
            |
            +-org.t_engine.tenet.core.CoreException
                |
                +-org.t_engine.tenet.core.CoreParameterException
```

public class **CoreParameterException**
extends `CoreException`

It is thrown, when the message argument is illegal.
For example, it is thrown when null has been specified for a parameter, for which null is not permitted.

Constructors

CoreParameterException

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

It structures a CoreParameterException that has a detailed message string.

Parameters:

`string` - Detailed message

org.t_engine.tenet.core

Class CoreSmartcardException

```
java.lang.Object
  |
  +-java.lang.Throwable
     |
     +-java.lang.Exception
        |
        +-org.t_engine.tenet.core.CoreException
            |
            +-org.t_engine.tenet.core.CoreSmartcardException
```

public class CoreSmartcardException
extends `CoreException`

It is thrown, when there is no IC card for the domain domain that the application accesses.

Constructors

CoreSmartcardException

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

It structures a CoreSCardException that has a detailed message string.

Parameters:

`string` - Detailed message

CoreSmartcardException

`public CoreSmartcardException(int errcode)`

It structures a CoreSmartcardException that has a detailed message string generated from the error code errcode

Parameters:

`string` - Error code

org.t_engine.tenet.core

Class DispatchAgent

```
java.lang.Object
  |
  +-org.t_engine.tenet.core.DispatchAgent
```

public class **DispatchAgent**
extends java.lang.Object

It provides the means for sending and receiving messages for application programs.
The instance of this class can be obtained by the SystemManager#getAgent() method. Each instance of the DispatchAgent class has a dynamically allocated, global unique eTRONID as the identifier. This identifier is used as the source ID and destination ID for message transmission. The allocated identifier value can be confirmed by the getIdentifier() method.
The application can send and receive e^{2}TP messages by using the acquired instance of this class. Two types of sending and receiving methods is provided for sending and receiving messages: synchronous message transmission (it awaits receipt of the answer message for the sent message; see syncSend()) and asynchronous message transmission (it returns immediately after a message transmission, without waiting for the receiving message; see send()).
To send a message, an AP invokes the send() or syncSend() method with a Message object as the argument. However, the eTRONID allocated when the instance is acquired is always used as the transmission source ID (srcID) of the message to be sent; even if a source ID is set in the argument object, it is always disregarded. When the thread ID of the argument object is undefined (null), the thread ID is allocated automatically (when it is not undefined, the thread ID set for the argument object is used as is). To assures global uniqueness of the thread ID, the allocated thread ID includes the sender's eTRONID (see "e^{2}TP message specifications" section 2.1.3)
In order to receive messages asynchronously, the application program must register a listener object for notifying the message receipt in advance, by the setListener() method. The listener object to be registered must implement the MessageListener interface. The message receipt is notified as an invocation of the MessageListener#received() method of the listener object. When no listener object has been registered, the received message is simply destroyed.

See Also:
  Message, MessageListener, SystemManager

Methods

getIdentifier

`public ETronID getIdentifier()`

It returns the eTRONID assigned to the DisptachAgent. This eTRONID is automatically used as the transmission ID, when sending messages.

Returns:
eTRONID assigned to the DispatchAgent.

setListener

`public void setListener(MessageListener listener)`

It records [t1]the listener 'listener' as the listener for receiving asynchronous messages. When a listener has previously been recorded, the specified listner overwrites it. When null is specified for 'listener', the previously registered listener is cleared.

Parameters:

`listener` – the Listener object to notify receipt of messages

---

getListener

`public MessageListener getListener()`

It acquires the listener recorded by the setListener() method. When nothing is recorded, it returns null.

Returns:

The listener recorded by the setListener() method

---

send

```
public ThreadID send(Message msg)
             throws CoreParameterException
```

It sends the message asynchronously to the address specified in the message 'msg'. When the sending of the message is complete, it immediately returns the control to the caller. It uses (overwriting) the eTRONID returned by the getIdentifier() method for the transmission source ID of the message msg to be sent, and if the thread ID of the message 'msg' to be sent is undefined (null), it sends the message using the newly generated thread ID. It returns the thread ID of the sent message.

Parameters:

`msg` – The message to be sent

Returns:

The thread ID of the message that was sent

Exceptions:

`CoreParameterException` – The message 'msg' to be sent is null

---

syncSend

```
public Message syncSend(Message msg)
                 throws CoreParameterException
```

It sends the message synchronously to the destination specified in the message 'msg'. This method does not return control until whether it receives the answer message corresponding to the sent message, i.e., the message which has the same thread ID as the sent message, or it becomes timeout. It returns the received message when the answer message is successfully received, or returns null if timeout. The length of the timeout period is specified by the SystemManager#getDefaultTimeout() method. Similar to the sync() method, the source ID in 'msg' is disregarded and the thread ID is allocated automatically if the thread ID of the 'msg' is undefined (null),

Parameters:

`msg` – The message to be sent

Returns:

The answer message to the sent message ‘msg’, or null if the reception of the answer message has timed out

Exceptions:

`CoreParameterException` – The message msg to be sent is null

---

syncSend

```
public Message syncSend(Message msg,
                         int timeout)
                  throws CoreParameterException
```

It synchronously sends the message to the destination specified by the message ‘msg’ , with explicit specification of the timeout period. This method is the same as the syncSend(msg) method, except the length of the timeout period is assigned by the argument.

Parameters:

`msg` – The message to be sent
`timeout` – The message receiving timeout (positive integer in milliseconds)

Returns:

When the answer message corresponding to the message to be sent has timed out, it is [t2]null

Exceptions:

`CoreParameterException` – The message ‘msg’ to be sent is null, or the message receiving timeout timeout is a negative number

org.t_engine.tenet.core

Class ETronID

```
java.lang.Object
  |
  +-org.t_engine.tenet.core.ETronID
```

public class **ETronID**
extends java.lang.Object

This class represents the eTRONID used as the identifier for indicating the message transmission source and destination for the e^2TP.

The eTRONID is composed of a domain and a port. This class provides, among others, the method for acquiring the eTRONID domain and port.

Constructors

ETronID

```
public ETronID(java.lang.String domain,
               int port)
```

Structures the ETronID that expresses the eTRONID composed of the domain character string expression (24 byte hex character string) domain and port.

Parameters:

`domain` - Domain character string expression (24 byte hex character string)
`port` - Port

Exceptions:

`CoreParameterException` – domain character string expression (24 byte hex character string) domain is null, or is not 24 bytes, or is not a hex character string.

ETronID

```
public ETronID(byte[] domain,
               int port)
```

Strcutures the ETronID object that expresses the eTRONID composed of the domain byte array expression domain and port.

Parameters:

`domain` – Domain byte array expression
`port` – Port

Exceptions:

`CoreParameterException` - Character string expression domain of the domain is null or other than 12 bytes

ETronID

`public ETronID(java.lang.String eTRONID)`

Structures the eTronID object that expesses the eTRONID, which is indicated by the eTRONID character string expression (32 byte hex character string) eTRONID

Parameters:

`eTRONID` - eTRONID character string expression (32 byte hex character string)

Exceptions:

`CoreParameterException` - eTRONID character string expression (32 byte hex character string) is null, or is other than 32 bytes, or is not a hex character string.

ETronID

`public ETronID(byte[] eTRONID)`

Structures the eTronID object that expresses the eTRONID indicated in the eTRONID byte array expression eTRONID.

Parameters:

`eTRONID` – eTRONID byte array expression

Exceptions:

`CoreParameterException` – The eTRONID byte array expression eTRONID is null or not 16 bytes

Methods

equals

`public boolean equals(java.lang.Object obj)`

Decides the eTRONID equivalency. When the eTRONID shown by the relative object obj is equivalent to the eTRONID of this object, true is returned. When eTRONID is different, or when the relative object ‘obj is not an instance of the ETronID class false is returned.

Parameters:

`obj` – Relative object

Returns:

When the eTRONID shown by the relative object ‘obj’ is equivalent to the eTRONID of this object, it is true, otherwise it is false

---

toString

`public java.lang.String toString()`

It returns this eTRONID character string expression (32 byte hex character string).

Returns:

eTRONID character string expression (32 byte array hex character string

---

toBytes

```
public byte[] toBytes()
```

It returns this eTRONID byte array.

Returns:

eTRONID byte array expression

getDomain

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

It returns this eTRONID domain character string expression (24 byte character string).

Returns:

Domain character string expression (24 byte character string).

getPort

```
public int getPort()
```

It returns this eTRONID port.

Returns:

Port

org.t_engine.tenet.core

Class Message

```
java.lang.Object
  |
  +-org.t_engine.tenet.core.Message
```

public class **Message**
extends java.lang.Object

This class is to represent the e^2TP message.

An instance of this class represents a message that includes the source ID (src), the destination ID (dest), the message type code (mtype), the thread ID (threadID), and the parameters (param).

The source ID cannot be specified as the argument of the constructors of this class; the source ID of the Message object generated by the constructor is always undefined. Similarly, the thread ID of the object generated by the constructor whose arguments are dest, mtype and param is undefined. This class provides the method for acquiring the source ID, the destination ID, the message type code, the thread ID and the parameters, and the method that returns the byte array expression of the message.

Constructors

Message

```
public Message(ETronID dest,
               int mtype,
               byte[] param)
```

It structures the message object that expresses the message structured from the destination ID dest, the message type code mtype and the parameter param. The thread ID is undefined for the message object generated by this constructor. When there is no parameter for the message to be constructed, null is to be specified as 'param'.

Parameters:

`dest` – Destination ID
`mtype` – Message type code
`param` – Parameter

Exceptions:

`CoreParameterException` – The destination ID dest is null.

Message

```
public Message(ETronID dest,
               int mtype,
               ThreadID threadID,
               byte[] param)
```

It structures the message object that expresses the message structured from the destination ID dest, the message type code mtype, the thread ID 'thread ID' and the parameter param. When there is no parameter for the message to be constructed, null is to be specified as 'param'.

Parameters:

`dest` – Destination ID
`mtype` – Message type code
`threadID` – Thread ID
`param` – Parameter

Exceptions:

`CoreParameterException` – The destination ID dest is null.

Methods

toBytes

`public byte[] toBytes()`

Returns the byte array expression of this message.

Returns:

Byte array expression of the message

---

getSrc

`public ETronID getSrc()`

Returns the transmission ID of this message. Returns null, when this transmission ID is undefined.

Returns:

Transmission ID

---

getDest

`public ETronID getDest()`

Returns the destination ID of this message.

Returns:

Destination ID

---

getMtype

`public int getMtype()`

Returns the message type code of this message.

Returns:

Message type code

---

getThread ID

`public ThreadID getThreadID()`

Returns the thread ID of this message. Returns null, when the thread ID is undefined.

Returns:

Thread ID

---

getParam

```
public byte[] getParam()
```

Returns the parameter of this message. Returns the 0 byte array, when the parameter is undefined.

Returns:

Parameter

org.t_engine.tenet.core

Interface MessageListener

public interface **MessageListener**

This interface is a interface for listeners which receives the answer message for the asynchronous message transmission

Registering the listeners implement this interface by the DispatchAgent#setListener() method enables APs to be notified of received messages.

`See Also:`
  `DispatchAgent`

Methods

received

`public Message received(Message msg)`

It is invoked when the DispatchAgent object has received a message. The argument msg is the message that was received. When the return value of this method is a Message object, the returned object is automatically sent as the message. In this case, the eTRONID of the DispatchAgent object associated with this object is used as the source ID of the message to be sent, and the same thread ID as the receiving message is also used automatically when the thread ID is undefined (null). When the return value is null or an object other than the message class, no message is sent automatically. When automatic sending of a message is not required, we recommend to return null as the return value.

Parameters:
`msg` – Received message

Returns:
A message to be sent, or null if no message need be sent in response

org.t_engine.tenet.core

Class SystemManager

```
java.lang.Object
  |
  +-org.t_engine.tenet.core.SystemManager
```

public class **SystemManager**
extends java.lang.Object

This class is a singleton class that initializes and sets the default values of the system.
It provides the method that inspects the name of the connected IC card reader-writers (IC card R/W), the domain map of the IC cards inserted in the IC card R/Ws and the default timeout for sending and receiving synchronous messages, and the method to qcquire the DispatchAgent objects providing the functions that send and receive messages for the application.

`See Also:`
`DispatchAgent`

Methods

getInstance

```
public static SystemManager getInstance()
                                 throws CoreInternalException
```

It initializes the environment required for sending and receiving e^{2}TP messages using this API and returns a unique instance (singleton) of this class.

Parameters:
The singleton instance of this class

Exceptions:
`CoreInternalException` – Failed accessing of resource (IC card R/W driver)

getDomainMap

```
public org.t_engine.tenet.util.Map getDomainMap()
```

It returns a Map object of the name of the smartcard R/Ws and the character string expression (24 byte hex character string) of the domain of the smartcards inserted into the R/Ws. The keys of the map are the names of the R/W.　When there is no usable IC card inserted into the IC card R/W, it returns an empty map.

Returns:
Map of the domain string expression (24 byte hex character string) of the IC card inserted into the IC card R/W keyed on the IC card R/W name

getDefaultTimeout

```
public int getDefaultTimeout()
```

Returns the default timeout for synchronous message transmission. The initial default timeout value depends on the environment

Returns:

The default timeout (positive integer in milliseconds)

---

setDefaultTimeout

```
public void setDefaultTimeout(int timeout)
                          throws CoreParameterException
```

Sets the default timeout for synchronous message sending and receiving.

Parameters:

`timeout` - The default timeout (positive integer in milliseconds)

Exceptions:

`CoreParameterException` - The default timeout timeout is a 1 negative number

---

getAgent

```
public DispatchAgent getAgent(java.lang.String domain)
                   throws CoreParameterException,
                          CoreErrorMessageException,
                          CoreSmartcardException
```

Instantiates and returns a DispatchAgent object whose eTRONID belongs to the 'domain'.

Parameters:

`domain` – The Domain that the instantiated DispatchAgent object belongs

Returns:

Instantiated DispatchAgent object

Exceptions:

`CoreParameterException` – The specified domain is not the correct domain character string expression (see eTronID)

`CoreErrorMessageException` – The IC card has failed to generate the port for the DispatchAgent object to be generated (The domain domain cannot generate any more new ports.)

`CoreSmartcardException` – The IC card corresponding to the specified domain does not exist in the environment.

org.t_engine.tenet.core

Class ThreadID

```
java.lang.Object
  |
  +-org.t_engine.tenet.core.ThreadID
```

public class **ThreadID**
extends java.lang.Object

The class represents the thread ID contained in the message.
This class provides the method that returns character string expression (40 byte hex character string) or byte array expression of the thread ID.

Constructors

ThreadID

`public ThreadID(byte[] threadID)`

It structures the thread ID object that shows the thread ID indicated by the byte string expression thread ID of the thread ID.

Parameters:
`thread ID` – Thread ID byte string expression

Exceptions:
`CoreParameterException` – Thread ID is null or not 20 bytes

Methods

toString

`public java.lang.String toString()`

It returns this thread ID character string expression (40 byte hex character string)

Returns:
Character string expression (40 byte hex character string) of the thread ID

toBytes

`public byte[] toBytes()`

Returns the ID byte array expression of this thread ID.

Returns:
Byte array expression of the thread ID.

Index

Appendix org.t_engine.util*

The authority value transaction API and e^{2}TP messaging API use 3 types of interfaces, org.t_engine.util.Set, org.t_engine.util.Map, org.t_engine.Iterator as the interfaces for handling the interface and iterator for handling both aggregate and associative arrays..

These interfaces respectively provide an interface of the same name present in the java.util package (java.util.Set, java.util.Map and java.util.Iterator) and the same interfaces except that the methods use org.t_engine.util.Collection instead of java.util.Collection as the argument or the return value; the org.t_engine.util.Collection interface provides the same interface as the java.util.Collection interface.

In other words, org.t_engine.util.* provides a subset of the JCF (Java Collections Framework), which is not provided in the J2ME CLDC environment.