      Voucher Trading System Application Programming Interface (VTS-API)
                <draft-ietf-trade-voucher-vtsapi-05.txt>

Status of This Document

 This document is an Internet-Draft and is in full conformance with
 all provisions of Section 10 of RFC2026.

 Internet-Drafts are working documents of the Internet Engineering
 Task Force (IETF), its areas, and its working groups. Note that
 other groups may also distribute working documents as Internet-
 Drafts.

 Internet-Drafts are draft documents valid for a maximum of six months
 and may be updated, replaced, or obsoleted by other documents at any
 time. It is inappropriate to use Internet-Drafts as reference
 material or to cite them other than as "work in progress."

     The list of current Internet-Drafts can be accessed at
     http://www.ietf.org/ietf/1id-abstracts.txt

     The list of Internet-Draft Shadow Directories can be accessed at
     http://www.ietf.org/shadow.html.

 Distribution of this document is unlimited. Please send comments to
 the TRADE working group at <ietf-trade@lists.elistx.com>, which may
 be joined by sending a message with subject "subscribe" to <ietf-
 trade-request@lists.elistx.com>.

 Discussions of the TRADE working group are archived at
 http://lists.elistx.com/archives/ietf-trade.

Abstract

   This document specifies the Voucher Trading System Application
   Programming Interface (VTS-API). The VTS-API allows a wallet or other
   application to issue, transfer, and redeem vouchers in a uniform
   manner independent of the VTS implementation. The VTS is a system to
   securely transfer vouchers, e.g., coupons, tickets, loyalty points,
   and gift certificates; this process is often necessary in the course
   of payment and/or delivery transactions.

Acknowledgements

   The following persons, in alphabetic order, contributed substantially
   to the material herein:

           Donald Eastlake 3rd
           Iguchi Makoto
           Yoshitaka Nakamura
           Ryuji Shoda

Table of Contents

M. Terada, K. Fujimura   [Page 2]

1. Introduction

   This document specifies the Voucher Trading System Application
   Programming Interface (VTS-API). The motivation and background of the
   Voucher Trading System (VTS) are described in Requirements for
   Generic Voucher Trading [VTS].

   A voucher is a logical entity that represents a certain right and is
   logically managed by the VTS. A voucher is generated by the issuer,
   traded among users, and finally collected using VTS. The terminology
   and model of the VTS are also described in [VTS].

   The VTS-API allows a caller application to issue, transfer, and
   redeem vouchers in a uniform manner independent of the VTS
   implementation. Several attempts have been made at providing a
   generic payment API. Java Commerce Client [JCC] and Generic Payment
   Service Framework [GPSF], for example, introduce a modular wallet
   architecture that permits diverse types of payment modules to be
   added as plug-ins and supports both check-like/cash-like payment
   models. This document is inspired by these approaches but the scope
   of this document is limited to the VTS model, in which cash-like
   payment model is assumed and vouchers are directly or indirectly
   transferred between sender (transferor) and receiver (transferee) via
   the VTS. This document is not intended to support API for SET,
   e-check or other payment schemes that do not fit the VTS model.

   Unlike the APIs provided in JCC and GPSF, which are designed to

transfer only monetary values, this API enables the transfer of a
wide-range of values through the use of XML-based Generic Voucher
Language [GVL]. The monetary meaning of the voucher is interpreted by
the upper application layer using the information described in the

   language. This approach makes it possible to provide a simpler API in
   the voucher-transfer layer and enhances runtime efficiency.
   The API specification in this document is described in the Java
   language syntax. Bindings for other programming languages may be
   completed in a future version of this document or separate related
   specifications.

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in
   this document are to be interpreted as described in [RFC 2119]

2. Processing Model

   This section provides the processing model in which the VTS-API is
   used. Most of the text in this section has been taken from the
   Generic Voucher Language specification [GVL].

   There are several ways of implementing VTS. For discount coupons or
   event tickets, for example, the smart-card-based decentralized
   offline VTS is often preferred, whereas for bonds or securities,
   the centralized online VTS is preferred. It is impractical to
   define standard protocols for issuing, transferring, or redeeming
   vouchers at this moment.

   To provide implementation flexibility, this document assumes a
   modular wallet architecture that allows multiple VTS to be added as
   plug-ins. In this architecture, instead of specifying a standard
   voucher transfer protocol, two specifications, i.e., Voucher
   Component and VTS-API specifications, are standardized (Figure 1).

```
   Sender wallet/Issuing system      Receiver wallet/Collecting system
   +--------------------------+      +--------------------------+
   |                          |      |                          |
   | |               Voucher Component               | |
   | |   (Specifies Issuer, Promise, Holder, and VTS Provider) | |
   | |---------------------------------------------------------->| |
   | |                          |      |                          | |
   | |          Intention to receive and payment (option)       | |
   | |<- - - - - - - - - - - - - - - - - - - - - - - - - - - - - | |
   | |                          |      |                          | |
   | |                          |      |                          | |
   | | Issue/transfer/  VTS     |      |  VTS      Register        | |
   | | redeem request   plug-in |      |  plug-in  Listener(*1)|   | |
   | |------------------>|       |      |       |<------------------|  |
   | | (VTS API)         |<- - - - - - - ->|         (VTS API) |   |
   | |                   | VTS-specific    |                      | |
   | |                   | protocol if VTS |                      | |
   | |                   | is distributed  |                      | |
   | |   Result          |<- - - - - - - ->|       Notify(*2)  |   |
```

```
   |   |<------------------|    |        |    |------------------>|   |
   +---------------------------+        +---------------------------+
   (*1) Registration is optional. Note also that the VTS plug-ins are
        usually pre-registered when the wallet or collecting system
```

    is started.

  (*2) If a listener is registered.

          Figure 1. Wallet architecture with VTS plug-ins

  After sender and receiver agree on what vouchers are to be traded and
  which VTS is to be used, the issuing system or wallet system requests
  the corresponding VTS plug-in to permit the issue, transfer, or
  redeem transactions to be performed via the VTS-API. The VTS then
  rewrites the ownership of the vouchers using the VTS-specific
  protocol. Finally, a completion event is sent to the wallet systems
  or issuing/collecting systems.

  This document describes the VTS-API specification. See [GVL] for the
  Voucher Component specification.

3. Design Overview

  We have adopted the following approach to specify the VTS-API.

  1) Provide an abstract and uniform API that encapsulates the VTS
     implementation. For example, a common API is provided for both
     centralized and decentralized VTS. It brings more freedom of VTS
     selection for issuers and application developers.

  2) To provide an abstract and uniform API, this document introduces
     an interface called VTSAgent that is associated with a holder and
     provides methods to manipulate vouchers held by its holder.
     Vouchers are accessed through the methods provided by the
     VTSAgent.

  3) Use existing standards for the VTS branding mechanism
     (negotiation). This document assumes that the VTS to be used for
     sending a voucher has settled before calling the VTS-APIs.
     Negotiation can be done within the upper application layer using
     other standards, e.g., [IOTP] or [ECML], if necessary.

  4) Support only push-type voucher transfer interface in which voucher
     transfer session is initiated by the transferor side. Pull-type
     voucher transfer interface can be implemented on top of the
     push-type VTS interface at application level.

4. Concepts

  The VTS-API consists of the following interfaces. A VTS is required
  to implement all of the interfaces except ReceptionLister, which is
  intended to be implemented by wallets or other applications that
  use VTS.

VTSManager

      Provides the starting point to use a VTS plug-in. All of the

objects needed to manipulate vouchers can be directly or indirectly
acquired via the VTSManager. A VTSManager maintains the two
repositories; a ParticipantRepository and a
VoucherComponentRepository described below.

ParticipantRepository

Provides the access points of Participants, which are to be trading
partners. A ParticipantRepository maintains Participants and acts as
an "address book" of trading partners.

Participant

Represents a participant (such as issuers, holders, and
collectors). A Participant knows how to obtain the corresponding
VTSAgent described below.

VTSAgent (extends Participant)

Provides the access point of vouchers in Valid Voucher Set (VVS)
that is logically managed by VTS. A VTSAgent provides a means of
manipulating vouchers held by its holder; basic trading methods,
i.e., issue, transfer, consume, and present. Before calling trading
methods, the application must create a Session which is described
below.

Session

Represents the logical connection established by the trade. A
Session has references to two Participants, i.e., the sender and the
receiver. After trading methods are called using a Session, the
Session holds a reference to the Vouchers to be traded.

Voucher

Represents one or more vouchers of which all of the issuer part and
promise part of vouchers are the same. A Voucher holds references to
the Participant (issuer) who issued the voucher and a
VoucherComponent (promise) which is described below.

VoucherComponent

Represents a Voucher Component described in [GVL]. It defines the
promise part of the voucher.

VoucherComponentRepository

Provides the access points of VoucherComponents. A
VoucherComponentRepository maintains VoucherComponents and acts as a
"voucher type book" managed by the VTS. This document assumes that a

set of VoucherComponents has been acquired and stored in this
repository. Delivery of VoucherComponents is beyond the scope of this
document. It may be delivered within the VTS from the trading

partners or manually acquired from a trusted third party (See Section
3 of [GVL]).

ReceptionListener

Provides a listener function with regard to the receipt of a voucher
by VTSAgent to wallets or other applications that implement this
interface. (This interface may not be implemented as part of VTS)

5. Interface Definitions

The interfaces defined in this document reside in the package named
"org.ietf.vts". Wallets or other applications that use this
API,should import this package as "import org.ietf.vts.*;".

5.1 VTSManager

  public interface VTSManager

  Provides the starting point to use a VTS plug-in.

  All of the objects needed to manipulate vouchers can be directly or
  indirectly acquired via a VTSManager, so that wallets or other
  applications can make the VTS available by instantiating an object
  implementing this interface.

  A class that implements the VTSManager interface must have a public
  default constructor (a constructor without any parameters). The VTS
  provides a name for such constructor so that the implementation class
  can bootstrap the interface.

5.1.1 getParticipantRepository

  public ParticipantRepository getParticipantRepository()

  Returns a repository that maintains Participants.

  Returns:

  the ParticipantRepository of the VTS, or null if no
  ParticipantRepository is available.

5.1.2 getVoucherComponentRepository

  public VoucherComponentRepository getVoucherComponentRepository()

  Returns a repository that maintains VoucherComponents.

  Returns:

the VoucherComponentRepository of the VTS, or null if no
VoucherComponentRepository is available.

[5.2](#) **ParticipantRepository**

  public interface ParticipantRepository

  Provides the access points of Participants. A ParticipantRepository
  maintains Participants and acts as an "address book" of trading
  partners.

  The object implementing this interface maintains Participants (or
  holds a reference to an object maintaining Participants), which are
  to be trading partners.

  The implementation of ParticipantRepository may be either (an adaptor
  to) "yellow pages" which is a network-wide directory service like
  LDAP, or "pocket address book" which maintains only personal
  acquaintances.

[5.2.1](#) **lookup**

  public Participant lookup(String id)

  Retrieves the participant that has the specified id.

  Returns:

  the participant associated with the specified id or null if the id
  is null or the corresponding participant cannot be found.

[5.3](#) **Participant**

  public interface Participant

  Represents the participants (such as issuers, holders, and
  collectors).

  This interface is used as representation of the trade partners and
  issuers of vouchers. Anyone can retrieve objects implementing
  Participant from the participant repository.

[5.3.1](#) **getIdentifier**

  public String getIdentifier()

  Returns the identifier of the participant. Each participant must
  have a unique identifier.

  The identifier can be used for looking up and retrieving the
  participant via the ParticipantRepository.

  The format of the identifier is implementation-specific.

Returns:

   the identifier string of the participant.

5.3.2 getVTSAgent

  VTSAgent getVTSAgent()

   Returns a VTSAgent, whose identifier is the same as the identifier of
   the participant.

   Returns:

   an object implementing VTSAgent.

5.4 VTSAgent

  public interface VTSAgent extends Participant

   Represents contact points to access vouchers in Valid Voucher Set
   (VVS) that is managed by the VTS.

   Each VTSAgent is associated with a holder and provides a means for
   managing vouchers owned by the holder. The holder must be
   authenticated using the login() method before being called by any
   other method, or VTSSecurityException will be issue.

   Before calling any trading method, i.e., issue(), transfer(),
   consume(), and present(), the application must establish a session by
   the prepare() method.

   Sessions may often be suspended due to network failure when the
   voucher is sent via a network. The suspended sessions can be
   restarted by the resume() method. Details on the state management of
   a session are described in Section 5.5.

   Some VTSAgents may not have all of the trading methods; a voucher
   collecting system doesn't require its VTSAgent to provide method for
   issuing or creating vouchers. A VTSAgent returns
   FeatureNotAvailableException when an unsupported method is invoked.

5.4.1 login

  public void login(String passphrase)
        throws VTSException

   Authenticates the VTSAgent. The passphrase is specified if the VTS
   requires it for authentication, otherwise it must be null. Nothing is
   performed if the VTSAgent has already been logged-in. The
   authentication scheme is implementation-specific. Examples of the
   implementation are as follows:

1) Vouchers are managed on a remote centralized server (central VVS),
      and the server requires a password to login. In this case, the
      application may prompt the user to input the password and can be

      given to the VTSAgent through this method.

   2) Vouchers are managed on a remote centralized server (central VVS);
      they require challenge-and-response authentication using
      smartcards held by users. In this case, the passphrase may be null
      since access to the smartcard can be done without contacting the
      application or user, i.e., the VTSAgent receives the challenge
      from the server, sends the challenge to the smartcard (within the
      VTS), and returns the response from the smartcard to the server.
      Note that a PIN to unlock the smartcard may be given through this
      method depending on the implementation.

   3) Each user holds their own smartcard in which their own vouchers
      are stored (decentralized VVS). In this case, the passphrase may
      be null since no authentication is required. Note that a PIN to
      unlock the smartcard may be given through this depends on the
      implementation.

   Throws:

    VTSSecurityException - if authentication fails.

5.4.2 logout

  public void logout()
         throws VTSException

   Voids the authentication performed by the login() method.

   After calling this method, calling any other method (except
   login()) will cause VTSSecurityException.

   The VTSAgent can login again by the login() method.

   Throws:

    VTSSecurityException - if the VTSAgent is not authenticated
      correctly.

5.4.3 prepare

  public Session prepare(Participant receiver)
         throws VTSException

   Establishes a session that is required for trading vouchers. The
   trading partner who receives the vouchers is specified as receiver.
   The vouchers to be traded will be specified later (when a trading
   method is called).

   The establishment of a session is implementation-specific. An

implementation that has a central VVS may start a transaction, while
other implementations that have decentralized VVS may get, from the
receiver, the challenge needed to authenticate the sender during the

establishment of the session.

If the VTSAgent has no ability to establish a session with the
specified receiver (permanent error), the VTSAgent throws an
InvalidParticipantExeption. If the VTSAgent can not establish a
session due to network failure (transient error), the VTSAgent throws
a CannotProceedException.

  Parameters:

   receiver - the trading partner who receives vouchers.

  Returns:

   an established session whose state is "prepared" (see Section 5.5).

  Throws:

   CannotProceedException - if the preparation of the session is
     aborted (e.g. network failures).
   FeatureNotAvailableException - if the VTSAgent does not provide
     any trading methods.
   InvalidParticipantException - if the specified participant is
     invalid.
   VTSSecurityException - if the VTSAgent cannot be authenticated
     correctly.

5.4.4 issue

  public void issue(Session session,
                    VoucherComponent promise,
                    java.lang.Number num)
         throws VTSException

   Issues vouchers. This method creates the specified number of
   vouchers <this, promise, receiver> and adds them to the VVS. Note
   that receiver is specified when the prepare() method is
   called. Nothing is performed if the specified number is 0.

   The session MUST be "prepared" when calling this method. The state
   of the session will be "activated" when the vouchers are created, and
   it will be "completed" when the transaction is successfully completed
   or "suspended" if the transaction is interrupted abnormally (e.g.,
   network failures).

  Parameters:

   session - the session used by the issue transaction.
   promise - the promise part of the voucher.
   num - the number of vouchers to be issued.

Throws:

```
CannotProceedException - if the transaction cannot be successfully
  completed.
FeatureNotAvailableException - if the VTSAgent does not provide
  a means of issuing vouchers.
InvalidStateException - if the session is not "prepared".
VTSSecurityException - if the VTSAgent cannot be authenticated
 correctly.
```

5.4.5 transfer

```
public void transfer(Session session,
                     Participant issuer,
                     VoucherComponent promise,
                     java.lang.Number num)
      throws VTSException
```

Transfers vouchers. This method rewrites the specified number of
vouchers <issuer, promise, this> to <issuer, promise, receiver> in
the VVS. Note that receiver is specified when the prepare() method is
called. The VTSAgent must have sufficient vouchers in the VVS.
Nothing is performed if the specified number is 0.

The session MUST be "prepared" when calling this method. The state
of the session will be "activated" when the voucher are retrieved
from the sender, and it will be "completed" when the transaction is
successfully completed or "suspended" if the transaction is
interrupted abnormally (e.g., network failures).

If null is specified for the issuer parameter, it indicates "any
issuer". This method selects vouchers to be transferred from the set
of vouchers returned by the getContents(null, promise).

 Parameters:

  session - the session used by the transfer transaction.
  issuer - the issuer part of the voucher, or null.
  promise - the promise part of the voucher.
  num - the number of vouchers to be transferred.

 Throws:

  CannotProceedException - if the transaction cannot be successfully
    completed.
  FeatureNotAvailableException - if the VTSAgent does not provide
    a means of transferring vouchers.
  InsufficientVoucherException - if the VTSAgent doesn't have a
    sufficient number of vouchers to transfer.
  InvalidStateException - if the session is not "prepared".
  VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.
```

### [5.4.6](#) consume

```
  public void consume(Session session,
                      Participant issuer,
                      VoucherComponent promise,
                      java.lang.Number num)
        throws VTSException
```

Consumes vouchers. This method deletes the specified number of
specified vouchers <issuer, promise, this> from the VVS. The VTSAgent
must have sufficient vouchers in the VVS. Nothing is performed if
the specified number is 0.

The session MUST be "prepared" when calling this method. The state
of the session will be "activated" when the vouchers are deleted, and
it will be "completed" when the transaction is successfully completed
or "suspended" if the transaction is interrupted abnormally (e.g.,
network failures).

If null is specified for the issuer parameter, it indicates "any
issuer". This method selects vouchers to be consumed from the set of
vouchers returned by the getContents(null, promise).

Parameters:

  session - the session used by the consume transaction.
  issuer - the issuer part of the voucher, or null.
  promise - the promise part of the voucher.
  num - the number of vouchers to be consumed.

Throws:

  CannotProceedException - if the transaction cannot be successfully
    completed.
  FeatureNotAvailableException - if the VTSAgent does not provide
    a means of consuming vouchers.
  InsufficientVoucherException - if the VTSAgent doesn't have a
    sufficient number of vouchers to consume.
  InvalidStateException - if the session is not "prepared".
  VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.

5.4.7 present

```
  public void present(Session session,
                      Participant issuer,
                      VoucherComponent promise,
                      java.lang.Number num)
        throws VTSException
```

Presents vouchers. This method shows that the sender has the
specified number of vouchers <issuer, promise, this> in the VVS to

the receiver of the session; No modification is performed to the
VVS. The VTSAgent must have a sufficient vouchers in the VVS. Nothing
is performed if the specified number is 0.

   The session MUST be "prepared" when calling this method. The state
   of the session will be "activated" when the vouchers are retrieved,
   and it will be "completed" when the transaction is successfully
   completed or "suspended" if the transaction is interrupted abnormally
   (e.g., by network failures).

   If null is specified for the issuer parameter, it indicates "any
   issuer". This method selects vouchers to be presented from the set
   of vouchers returned by the getContents(null, promise).

  Parameters:

   session - the session used by the present transaction.
   issuer - the issuer part of the voucher, or null.
   promise - the promise part of the voucher.
   num - the number of the voucher to be presented.

  Throws:

   CannotProceedException - if the transaction cannot be successfully
     completed.
   InsufficientVoucherException - if the VTSAgent doesn't have a
     sufficient number of vouchers to present.
   InvalidStateException - if the session is not "prepared".
   FeatureNotAvailableException - if the VTSAgent does not provide
     a means of presenting vouchers.
   VTSSecurityException - if the VTSAgent cannot be authenticated
     correctly.

5.4.8 cancel

  public void cancel(Session session)
        throws VTSException

   Releases the session. "Prepared" sessions MUST be canceled. An
   implementation MAY be permitted to cancel "activated" or "suspended"
   sessions.

  Throws:

   InvalidStateException - if the state of the session isn't
     cancelable.
   VTSSecurityException - if the VTSAgent cannot be authenticated
     correctly.

5.4.9 resume

  public void resume(Session session)
        throws VTSException

Restarts the session. Only "suspended" sessions can be resumed.

The state of the session will be re-"activated" immediately, and it
will be "completed" when the transaction is successfully completed or
"suspended" again if the transaction is interrupted abnormally (e.g.,
network failures).

Throws:

CannotProceedException - if the transaction cannot be successfully
    completed.
InvalidStateException - if the session is not "suspended".
VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.

5.4.10 create

```
public void create(VoucherComponent promise, java.lang.Number num)
        throws VTSException
```

Creates vouchers where the issuer is the VTSAgent itself. This
method creates the specified number of vouchers <this, promise,
this> and adds them to the VVS. Nothing is performed if the
specified number is 0.

Throws:

FeatureNotAvailableException - if the VTSAgent does not provide
    a means of creating vouchers.
VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.

5.4.11 delete

```
public void delete(Participant issuer,
                   VoucherComponent promise,
                   java.lang.Number num)
        throws VTSException
```

Deletes vouchers. This method deletes the specified number of
vouchers <issuer, promise, this> from the VVS. The VTSAgent must
have sufficient vouchers in the VVS. Nothing is performed if the
specified number is 0.

Throws:

InsufficientVoucherException - if the VTSAgent doesn't have
    sufficient number of vouchers to delete.
VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.

5.4.12 getContents

```
   public java.util.Set getContents(Participant issuer,
                                    VoucherComponent promise)
```

```
        throws VTSException
```

  Returns the set of vouchers whose issuer and promise both match the
  issuer and promise specified in the parameters.

  If null is specified for the issuer or promise parameter, it
  indicates "any issuer" or "any promise", respectively. If null is
  specified for both parameters, this method selects all vouchers owned
  by the holder from the VVS.

 Returns:

  the set of vouchers held by the holder of the VTSAgent.

 Throws:

  VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.

5.4.13 getSessions

```
  public java.lang.Set getSessions()
          throws VTSException
```

  Returns a set of not-completed sessions prepared by the VTSAgent.

 Returns:

  the set of sessions prepared by the VTSAgent and not yet completed.

 Throws:

  VTSSecurityException - if the VTSAgent cannot be authenticated
    correctly.

5.4.14 getLog

```
  public java.lang.Set getLog()
          throws VTSException
```

  Returns a set of completed sessions prepared or received by the
  VTSAgent. This set represents the trading log of the VTSAgent. A
  VTS may delete an old log eventually, so that the entire log may
  not be returned; the amount of the log kept by the VTSAgent is
  implementation-specific.

 Returns:

  the set of completed sessions prepared or received by the VTSAgent.

Throws:

   VTSSecurityException - if the VTSAgent cannot be authenticated

       correctly.

5.4.15 addReceptionListener

  public void addReceptionListener(ReceptionListener l)
          throws VTSException

   Adds a ReceptionListener to the listener list.

   After a ReceptionListener l is registered by this method, l.arrive()
   will be called whenever the VTSAgent receives a voucher.

   Nothing is performed if the specified listener is null.

  Throws:

   VTSSecurityException - if the VTSAgent cannot be authenticated
     correctly.

5.4.16 removeReceptionListener

  public void removeReceptionListener(ReceptionListener l)
          throws VTSException

   Removes a ReceptionListener from the listener list.

   Nothing is performed when the specified listener is null or not
   registered.

  Throws:

   VTSSecurityException - if the VTSAgent cannot be authenticated
     correctly.

5.5 Session

  public interface Session

   Represents the logical connection established by the trade. Sessions
   are established by VTSAgent#prepare().

   A session has four states: prepared, activated, suspended, and
   completed. The initial state of a session is "prepared", and the
   session will be "activated" immediately when any of the trading
   methods of VTSAgent is called. The "activated" session will be
   "completed" after the trading method is successfully completed. If
   the trading method is transiently failed (e.g. network failure), the
   session will be "suspended". Suspended sessions can be re-"activated"
   and restarted by calling VTSAgent#resume().

A completed session may disappear from the VTSAgent; the session
will be collected by the GC unless other objects keep its reference.

5.5.1 getIdentifier

  public String getIdentifier()

  Returns the identifier of the session. The generation scheme of the
  identifier is implementation-specific. An implementation may use a
  transaction ID as the identifier of the session.

  Returns:

  the string of the identifier of the session.

5.5.2 getVoucher

  public Voucher getVoucher()

  Returns the voucher to be traded using the session, or returns null
  if the session has not been activated.

  Returns:

  the voucher to be traded or null if the state of the session is
  "prepared".

5.5.3 getSender

  public Participant getSender()

  Returns the sender of the session, i.e., the creator who prepared the
  session.

  Returns:

  the sender of the session.

5.5.4 getReceiver

  public Participant getReceiver()

  Returns the receiver of the session, i.e., the participant specified
  when preparing the session (by the VTSAgent#prepare() method).

  Returns:

  the receiver of the session.

5.5.5 isPrepared

  public boolean isPrepared()

  Verifies if the session is "prepared".

Returns:

   true if the session is in "prepared" state, or false.

5.5.6 isActivated

  public boolean isActivated()

   Verifies if the session is "activated".

   Returns:

   true if the session is in "activated" state, or false.

5.5.7 isSuspended

  public boolean isSuspended()

   Verifies if the session is "suspended".

   Returns:

   true if the session is in "suspended" state, or false.

5.5.8 isCompleted

  public boolean isCompleted()

   Verifies if the session is "completed".

   Returns:

   true if the session is in "completed" state, or false.

5.6 Voucher

  public interface Voucher

   Represents voucher(s) described in [VTS]. An object implementing
   this interface can represent more than one voucher if all of the
   issuer part and the promise part of the vouchers are the same.

5.6.1 getIssuer

  public Participant getIssuer()

   Returns the issuer part of the voucher(s).

   Returns:

   the participant who issued the voucher(s).

5.6.2 getPromise

  public VoucherComponent getPromise()

   Returns the promise part of the voucher(s).

  Returns:

   the voucher component that defines the promise of the voucher.

5.6.3 getCount

  public java.lang.Number getCount()

   Returns the number of the voucher(s).

  Returns:

   the positive (>0) number of the voucher(s).

5.7 VoucherComponentRepository

  public interface VoucherComponentRepository

   Maintains VoucherComponents.

   An object implementing VoucherComponentRepository provides a means of
   retrieving the voucher components that are the promises of vouchers
   in the VVS.

   Before issuing a voucher, the promise of the voucher must be
   registered with this repository. The repository can be implemented
   as either a network-wide directory service or personal storage like
   the ParticipantRepository.

5.7.1 register

  public VoucherComponent register(org.w3c.dom.Document document)

   Creates a voucher component associated with the specified DOM object
   and registers the voucher component with the repository.

   A voucher component of the voucher to be issued must be registered
   using this method.

   Nothing is performed (and the method returns null) if the specified
   document is null or the syntax of the document does not conform to
   the VTS.

   The method returns the registered voucher component if the specified
   DOM object has been already registered. (No new voucher component is
   created in this case).

Returns:

     a registered voucher component associated with the
     specified document, or null if the document is null or has wrong
     syntax.

5.8 VoucherComponent

  public interface VoucherComponent

   Represents the voucher component that defines the promise of the
   voucher.

   Each VoucherComponent object has its own unique identifier, and it is
   associated with an XML document that describes the promise made by
   the issuer of the voucher, e.g., the goods or services can be claimed
   in exchange for redeeming the voucher.

   This interface can be implemented as sort of a "smart pointer" to the
   XML document. An implementation may have a reference to a voucher
   component repository instead of the voucher component and retrieve
   the document dynamically from the repository when the getDocument()
   method is called.

5.8.1 getIdentifier

  public String getIdentifier()

   Returns the identifier of the voucher component. Each voucher
   component must have a unique identifier. The identifier may be
   used to check for equivalence of voucher components.

   The format of the identifier is implementation-specific, however, it
   is RECOMMENDED to include the hash value of the voucher component in
   the identifier to assure its uniqueness. For generating the hash
   value, it is desirble to use a secure hash function, e.g., [SHA-1],
   and to apply a canonicalization function, e.g., [EXC-C14N], before
   applying the hash function to minimize the impact of insignificant
   format changes to the voucher component, e.g., line breaks or
   character encoding.

  Returns:

    The identifier string of the voucher component.

5.8.2 getDocument

  public org.w3c.dom.Document getDocument()

   Returns a Document Object Model [DOM] representation of the document
   associated with the voucher component by the
   VoucherComponentRepository#register() method.

The DOM object to be returned may be retrieved from a
VoucherComponentRepository on demand, instead of the VoucherComponent

   always keeping a reference to the DOM object.

   The VTS must guarantee that the getDocument method will eventually
   return the DOM object provided that the voucher associated with the
   corresponding voucher component exists in the VVS.

  Returns:

   a DOM representation of the document associated with the voucher
   component.

  Throws:

   DocumentNotFoundException - if the associated DOM object cannot be
      retrieved.

5.9 ReceptionListener

  public interface ReceptionListener extends java.util.EventListener

   Provides a listener interface that provides notification that a
   VTSAgent has been received a voucher.

   When a voucher arrives at VTSAgent, the VTSAgent invokes arrive()
   method of each registered ReceptionListener. ReceptionListeners can
   obtain a Session object, which contains information about the
   received voucher and the sender of the voucher.

   This interface is intended to provide a means of notifying a wallet
   that "You have new vouchers", so that this interface may be
   implemented by wallets or other applications using VTS.

5.9.1 arrive

  public void arrive(Session session)

   Provides notification of the arrival of a voucher.

   After the listener is registered to a VTSAgent (by the
   VTSAgent#addReceptionListener() method), the VTSAgent invokes this
   method whenever it receives a voucher.

   The specified session is equivalent to the session used by the sender
   to trade the voucher. The state of the session is "completed" when
   this method is called.

5.10 Exceptions

java.lang.Exception
  +-- VTSException

```
        +-- CannotProceedException
        +-- DocumentNotFoundException
        +-- FeatureNotAvailableException
```

```
   +-- InsufficientVoucherException
   +-- InvalidParticipantException
   +-- InvalidStateException
   +-- VTSSecurityException
```

VTSException

 This is the superclass of all exceptions thrown by the methods in the
 interfaces constructs the VTS-API.

CannotProceedException

 This exception is thrown when a trading is interrupted due to
 network failures or other errors.

DocumentNotFoundException

 This exception is thrown when the document associated with a voucher
 component cannot be found.

FeatureNotAvailableException

 This exception is thrown when the invoked method is not supported.

InsufficientVoucherException

 This exception is thrown when the number of the voucher is less than
 the number specified to trade.

InvalidParticipantException

 This exception is thrown when the specified participant cannot be
 located.

InvalidStateException

 This exception is thrown when the state of the session is invalid to
 proceed the operation.

VTSSecurityException

 This exception is thrown when authentication fails or a method
 which requires authentication in advance is called without
 authentication.

6. Example Code

 // Issue a voucher

 VTSManager vts = new FooVTSManager();

```
ParticipantRepository addrBook = vts.getParticipantRepository();
VoucherComponentRepository vcr = vts.getVoucherComponentRepository();
```

```
  Participant you = addrBook.lookup("http://foo.bar/baz");
  VTSAgent me = addrBook.lookup("myName").getVTSAgent();

  VoucherComponent promise = vcr.register(anXMLVoucherDocument);

  try {
    me.login();
    s = me.prepare(you);
    me.issue(s, promise, 1);
    me.logout();
  } catch (VTSException e) {
    System.err.println("Sorry!");
    e.printStackTrace();
  }

  // Transfer all my vouchers

  VTSManager vts = new FooVTSManager();
  ParticipantRepository addrBook = vts.getParticipantRepository();

  Participant you = addrBook.lookup("http://foo.bar/baz");
  VTSAgent me = addrBook.lookup("myName").getVTSAgent();

  try {
    me.login();
    Iterator i = me.getContents(null, null).iterator();

    while (i.hasNext) {
      Voucher v = (Voucher) i.next();
      s = me.prepare(you);
      me.transfer(s, v.getIssuer(), v.getPromise(), v.getCount());
    }

    me.logout();
  } catch (VTSException e) {
    System.err.println("Sorry!");
    e.printStackTrace();
  }

  // Register an incoming voucher notifier (biff)

  VTSManager vts = new FooVTSManager();

  ParticipantRepository addrBook = vts.getParticipantRepository();
  VTSAgent me = addrBook.lookup("myName").getVTSAgent();

  ReceptionListener listener = new ReceptionListener() {
    public void arrive(Session s) {
      System.out.println("You got a new voucher.");
    }
```

```
      };

      try {
```

```
     me.login();
     me.addReceptionListener(listener);
     me.logout();
   } catch (VTSException e) {
     System.err.println("Sorry!");
     e.printStackTrace();
   }
```

7. Security Considerations

This document assumes that the VTS plug-in is trusted. The caller
application of a VTS should authenticate the VTS plug-in and bind it
securely using the VTS Provider information specified in the Voucher
Component. This document, however, does not specify any application
authentication scheme and it is assumed to be specified by other
related standards. Until various VTS systems are deployed, it is
enough to manually check and install VTS plug-ins like other download
applications.

To protect vouchers from being stolen, the VTSAgent must be
authenticated securely. This document introduced a login/logout
facility for this purpose (see Section 5.4).

8. Normative References

[DOM] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le
Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood.
"Document Object Model (DOM) Level 1 Specification", W3C
Recommendation, October 1998,
<http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>

[GVL] K. Fujimura and M. Terada, "XML Voucher: Generic Voucher
Language", draft-ietf-trade-voucher-lang-05.txt, 2003.

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, 1997.

9. Informative References

[ECML] J. W. Parsons and D. Eastlake "Electronic Commerce Modeling
Language (ECML) Version 2 Specification",
draft-ietf-trade-ecml2-spec-08.txt, 2003.

[EXC-C14N] J. Boyer, D. Eastlake, and J. Reagle, "Exclusive XML
Canonicalization Version 1.0", W3C Recommendation, July 2002,
<http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>

[GPSF] G. Lacoste, B. Pfitzmann, M. Steiner, and M. Waidner (Eds.),
"SEMPER - Secure Electronic Marketplace for Europe," LNCS 1854,
Springer-Verlag, 2000.

   [IOTP] D. Burdett, "Internet Open Trading Protocol - IOTP Version
   1.0", RFC 2801, 2000.

[JCC] Sun Microsystems Inc., "Java Commerce Client",
<http://java.sun.com/products/commerce/>.

[SHA-1] Department of Commerce/National Institute of Standards and
Technology, "FIPS PUB 180-1. Secure Hash Standard. U.S.",
<http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>

[VTS] K. Fujimura, "Requirements and Design for Voucher Trading
System (VTS)", draft-ietf-trade-drt-requirements-04.txt, 2002.
In RFC Editor queue.

10. Author's Address

Masayuki Terada and Ko Fujimura
NTT Corporation
1-1 Hikari-no-oka, Yokosuka-shi, Kanagawa, 239-0847 JAPAN
Phone: +81-(0)46-859-3814
Fax: +81-(0)46-859-8329
Email: terada@isl.ntt.co.jp, fujimura@isl.ntt.co.jp