Data Transport Standard

National Council of Higher Education Loan Programs

HomePage

This site is a [Wiki]. A "wiki" is a website that is collaboratively edited by its users, including the ability to change text written by other users. Yes, that means that everyone, including you, has the ability to modify every single page on this site.

The Data Transport Standard (DTS) is a Web Services based transport system that is able to support many business data exchange needs. Initially designed to support synchronous and asynchronous transport models, it is possible to support batch, near-time and real-time needs within this framework. Additionally, current designs in progress include two different client configurations: both participants have client software and server software, one participant has client software and utilizes that software to communicate with a participant's server.

DTS utilizes SOAP, HTTP, SSL, XML, Base64, zLib compression, and UUID. The protocol uses SOAP over HTTPS to ensure the privacy of the transmission. XML is used to describe routing information related to a transmission, allowing data transport systems to be focused on moving data with little specific knowledge about the information contained within the transmission. UUID is utilized to generate unique message tracking numbers to ensure easier identification of specific messages. Base64 and zLib are used to compress (zlib) and encode (Base64) the data to be moved. This makes a DTSP system payload insensitive, capable of moving any type of data between business partners.

The purpose of this group is to recommend a business direction for “Data Transport” and define the business requirements for that process that can be used across multiple business sectors supported by Guarantors, Lenders, Schools, FAMS vendors and FSA.

Data Transport Business Workgroup

Business Requirements

upload:Proposed%20Transport%20Standard.doc

Stack:

- Application
- Guaranteed Delivery
- Security

    * Authorization
    * Authentication
    * Encryption

- [Core]

    * Coretesters
        * Data Type WSDL: [DTSDataTypes.wsdl]
    * Sample Service (getStatus): [getStatus.wsdl]
    * Endpoint for getStatus: [dts.wsdl]

Guaranteed Delivery

See the TaskList

There is a working group at OASIS that is working on Web Serivces - Reliable Messaging (WS-RM). The main page for this proposal is at [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm]. The message archives have a lot of pertinent information.

TaskList

This is the list of tasks (and hopefully dates) that need to be completed for the Data Transport Standard. If you see a task that you're interested in, please feel free to add your name as the responsible person.

- JAD location and date. - by 3/19/04

- Create Core specification documentation

    - Generate WSDL for a sample core level transport service - Mark and Tim will provide corrections/suggestions by 3/25/04. Mark will call Tim at 10:30 EST 3/19/04
        - Generate the WSDL for CR:C
        - Generate instructions for new types of data
    - Complete documentation for the core specification - John Gill will check with Gary and Kim for changes with modifications done by 3/25/04
    - Review and approve the core specification documentation - EEAT
    - Document the best practices for Java with Apache Axis and .Net with Microsoft SOAP - This will be an end result of the JAD in April
        - Take the best practices to PESC for approval

- Create Security specification documentation
    - Complete documentation for the security specification - Zahida will send to the group by 3/24/04
    - Review and approve the security specification documentation - Group will review on 3/25/04
    - Prototype sample service with the security layer - April JAD
        - Specify which certificates will be necessary for sample testing - Tim will verify 3/25/04
    - Document the best practices for Java and .Net - April JAD
    - Key management and centralized directory - Zahida will enhance the documentation for review by the groupby 3/24/04
        - This will be simililar to the Meteor centralized registry and we can use much of their documentation
        - Who will host the registry
        - Need procedures for updating the registry

- Create the Guaranteed Delivery specification documentation - John and an item for the JAD session
    - Complete documentation for the guaranteed delivery layer
    - Review and approve the documentation
    - Document the best practices for Java and .Net
        - How are files managed?
        - What is the base functionality that must be exposed?
    - Automatic retry
        - Number of times to retry, how often?
        - What happens when retry fails?
        - How does this relate or differentiate for real-time versus delayed processing

- Define large message transport requirements and limitations - JAD and EEAT the week following the JAD to be completed by 4/15/04
    - What are the implications for large file sizes and what hardware and bandwidth are required for various sized files

- Implementation
    - Documentation for implementing sample service with the core WSDL. Both subtasks will need rudimentary documentation to be available before the JAD
        - Java documentation - Mark and Tim
        - .Net documentation - Nathan and Zahida
    - Implement a test-bench representing the sample service - April JAD
    - Create standard sample files of varying sizes - Mark and Nathan will provide by 3/25/04

- Error handling definition and procedures
    - What happens when we receive a file to the incorrect destination

JAD Tasklist

- Create Core specification documentation
    - Document the best practices for Java with Apache Axis and .Net with Microsoft SOAP - This will be an end result of the JAD in April

- Create Security specification documentation
    - Prototype sample service with the security layer - April JAD
    - Document the best practices for Java and .Net - April JAD

- Create the Guaranteed Delivery specification documentation - John and an item for the JAD session
    - Complete documentation for the guaranteed delivery layer
    - Review and approve the documentation
    - Document the best practices for Java and .Net
        - How are files managed?
        - What is the base functionality that must be exposed?
    - Automatic retry
        - Number of times to retry, how often?
        - What happens when retry fails?

        - How does this relate or differentiate for real-time versus delayed processing

- Define large message transport requirements and limitations - JAD and EEAT the week following the JAD to be completed by 4/15/04
    - What are the implications for large file sizes and what hardware and bandwidth are required for various sized files

- Implementation
    - Implement a test-bench representing the sample service - April JAD

Security

We really need to watch [JSR 183] for Web Services Sercurity.
If we're going to use SAML assertions we should watch [JSR 155]
This site has several articles on security topics relating to Web Services : [http://webservices.xml.com/security]
Some resources on Web Service Security:
Microsoft's [Web Services Security Specifications Index Page] and IBM's [WS-Security Home Page] which both cover the exact same specification.
[Web Services Discovery]

Data Transport Core Standard

This is a protocol specification only and must be easily implementable in Java and .Net.

We are using a layered approach to building the protocol. The core protocol is a specification for transmitting data from one point to another. Authentication, authorization and automatic retransmission are handled in higher level specifications.

The body of the SOAP message will include exactly one element named "payload" with zero attributes which is a Base 64 encoded, Zlib compressed CDATA section.

Each request and response must contain the following header elements (http://www.datatransportstandard.com/uploads/DTSDataTypes.wsdl):

Sender - Type core:EntityType?
Recipient - Type core:EntityType?
TransportUUID? - UUID of type xs:String
TransmitDateTimeGMT? - Type xs:dateTime

Any implementation of the core protocol must be able to handle additional unknown header elements.

Any request that is received with any of these header elements missing will be rejected and a SOAP Fault will be returned.

Every distinct file type will have a standard SOAP method name to use. The method name will closely correlate to the Email and FTP filenames currently defined by ESC.

All method names will follow the following naming convention:

- [submit/request] - submit is used when pushing data to the trading partner. request is used when requesting the trading partner to return data.
- [File Type] - CRC, CAM, CL. All Upper Case
- [Version] - two digit number with the version of file being sent/requested
- [File Sub Type] - Camel Case of the current sub type
- {Batch} - If the response is not expected immediately then this request is to be considered a "Batch Mode" request. If the response is expected in real-time, then no suffix will be used.

As an example, CRC Version 1 Application Send request expecting a real-time response will be named submitCRC01AppSend?. A similar request with an expected response at a later time would be named submitCRC01AppSendBatch?

There will be a centralized LDAP server to act as a registry for all SOAP methods supported. This centralized registry will contain each participating entity. Each entity will contain business and technical contact information. A list of all supported transaction types and the public X.509 key of the entity will also be stored.

There are some initial requirements of the Data Transport Standard that are outside the scope of the core protocol. Since they do not

affect every file type send, they will be handled on a case by case basis in the application layer.

- Requirement: If you receive a change for a wrapper out of order then reject the first.

This is an application level requirement. Most transaction types do not need this requirement.

Stack:

- Application
- Guaranteed Delivery
- Security

  * Authorization
  * Authentication
  * Encryption

- Core

Development

There are many parallel tasks occuring. They are documented on the CodeDevelopment page.

Here are the basic tools needed to participate in effort. Toolkit

TaskList - The list of tasks that need to be completed.

Reevaluation of HPCP

upload:DTS%20overview.doc

upload:DTS%20core.doc

Current codebase as of 3/11/04: upload:dtsClient.zip
http://www.datatransportstandard.com/cgi/wiki.cgi?TOC

Re-evaluation of the Technical Specification of the High Performance Channel Protocol

Tim Bornholtz, PTI
John Gill, TG

Purpose of High Performance Channel

The purpose of the High Performance Channel Protocol Specification (HPCP) is to define standard mechanisms to securely transport data between members of the FFELP industry in real-time or synchronously. This specification is a usage guide for a set of related technologies. At its core, an implementation of the HPCP transmits data from one provider to another. The receiver of the data then responds appropriately to the sender depending on the request sent. The request sent may be binary, text as in CommonLine?, XML, or any other data format. Likewise, the response can be any data format. In addition to the actual data sent, the HPCP request and response contains other information necessary to help the receiver determine the validity of the data, the identity of the sender, and the action or actions to take with the data. The HPCP Specification also describes how data is to be encoded during transmission.

Disadvantage of the Current Protocol Specification

The current protocol specification is designed to be a layer on top of SOAP. We believe that this is a fundamental flaw in the HPCP technical specification. By looking at the abstract of SOAP we see that the goals of SOAP are nearly identical to the goals of HPCP.

SOAP Abstract (http://www.w3.org/TR/SOAP)

> SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

A better approach would be to remove the cumbersome and unnecessary layer that exists today, using SOAP as the basis of the transport, with the HPCP specification to define additional pieces of information within the SOAP message.

By comparing a simple SOAP message with a simple HPCP message the similarities become obvious.

```
SOAP Message
   <SOAP-ENV:Envelope/>
     <SOAP-ENV:Header>
          { Any custom header tags necessary such as... }
          <t:Transaction  xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
               5
          </t:Transaction>
     </SOAP-ENV:Header>
     <SOAP-ENV:Body>
               { XML Content Here}
     </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
HPCP Message
  <Envelope>
     <Message ID="{uuid}" TimeStamp?="{ccyymmddhhnnssttt}"/>
     <Sender ID="ED.{ED/DOE/NCHELP assigned code}"/>
     <Receipent ID="ED.{ED/DOE/NCHELP assigned code}"/>
     <Transaction Type="{send,resp,ack,err}"/>
     <Package>
          <Special Type="SAMPLEREQUEST" Encoding="NONE" Compression="NONE">
               <![CDATA[ {Optional Special Payload data} ]]>
          </Special?>
          <Content Type="SAMPLEREQUEST" Encoding="NONE" Compression="NONE">
               <![CDATA[ {Content Payload data} ]]>
          </Content?>
     </Package?>
  </Envelope?>
```

As you can easily see from these simple examples, the SOAP format and the HPCP format are very similar. Both contain an outer Envelope to enclose the entire message. Both contain information that is helpful to the processing of the actual body/content. Both send the actual data, SOAP through the <SOAP-ENV:Body> tag and HPCP through the <Content> tag. The current protocol specification wraps the content in an HPCP message structure and then wraps that entire HPCP message in a SOAP message. This additional wrapping of the content is redundant and provides no benefit to the data transport not already provided by SOAP. If the protocol specification is changed to utilize the SOAP message as the transport layer and define specific <SOAP-ENV:Header> elements to ensure consistent processing of the request, there are many benefits to be gained.

1) Speed Any implementation of SOAP must parse the entire request. This includes the portion of the request that is specific to HPCP. An implementation of HPCP must also parse and process their entire request. By merging the HPCP structure into the SOAP message, the entire contents will only need to be parsed and processed once.

2) Extensibility and Standards Conformance Most new XML standards that we are interested in using are developing with SOAP in mind. Most toolkits that implement various standards can already interact with the SOAP message with very little custom development necessary from an implementation of the HPCP. The current protocol specification is completely custom and does not follow W3C standards. Because of this, any additional standards that are to be implemented within a HPCP implementation require a great deal of development effort to implement.

3) Tool Integration Many tools in use today can process or create SOAP message with absolutely no development necessary. Since these tools understand the core SOAP specification, they can generate the necessary information with a simple Web Services Description Language (WSDL) file. Using the current implementation of HPCP, it is not possible to develop a WSDL file that is of any practical use.

4) Simplicity Because the proposed changes to the HPCP specification would conform to the SOAP standard and additional toolkits for securing the transaction are pure standards based, an implementation should be a simple task. An open source implementation of the specification can be created by the EEAT and be made available to the community. Some organizations, however, may wish to create their own implementation of the specification. These custom implementations should be simple to create and simple to test for interoperability with the reference implementation. Securing HPCP The OSI defines six basic security services to secure communications. The protocol specification must ensure that these security services are correct above all else. 1) Authentication 2) Access Control (Authorization) 3) Data Confidentiality 4) Data Integrity 5) Non-repudiation 6) Logging and Monitoring

Using a "best practices" approach to secure the protocol, we can be confident that all implementations of the protocol specification are indeed secure.

The following technologies can be used in conjunction with each other to secure the transmission.

Security Assertion Markup Language (SAML) is an XML based framework for ensuring that transmitted communications are secure. SAML defines mechanisms to exchange authentication, authorization, and non-repudiation information.

SOAP Security Extensions: Digital Signature (http://www.w3.org/TR/SOAP-dsig/) is a standard to specify the syntax and processing rules of a SOAP Header to carry XML Signature (http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/) information.

Centralized UDDI or LDAP registry to authorize senders and recipients, perform key management and maintain URL resources. The information stored in the registry contains only non-sensitive public information.

Secure Socket Layer (SSL) encrypts all communication between two endpoints.

Looking at the six security services, we can see how changes to the HPCP specification can ensure secure implementations.

1) Authentication SAML Assertion in the SOAP-Env:Header that contains sender and recipient. The sender's ID will be used to look up their public key to validate the Body of the message. If the Body validation fails, the request fails.

2) Access Control (Authorization) Each participant will have the authorized transaction types stored in a registry. This includes sending as well as receiving transactions. If a transaction is received for an entity that is not allowed to make the request, the transaction will fail.

3) Data Confidentiality All communications on the network are secured with SSL. As implementations of the XML Encryption standard emerge, they may be allowed to be used in lieu of or in conjunction with SSL.

4) Data Integrity The message body is signed with the signature sent in the header of the message. If the data has changed in any way (including whitespace) then the signature validation will fail and the transaction will fail.

5) Non-repudiation A required element of the SOAP-Env:Header is a Universally Unique Identifier (UUID). This identifier is guaranteed to be unique. This identifier can be logged and later be used in conjunction with the XML Signature to track particular requests.

6) Logging and Monitoring It is the responsibility of each implementation of the HPCP specification to log the information

necessary to track any given request for a fixed amount of time. Each installation should monitor their own network for suspicious activity. Specific logging messages and monitoring frameworks are outside of the scope of the protocol specification. In fact, if a specific monitoring specification were to be required in the protocol specification, the security could be greatly compromised.

Conclusion

Rather than build a complete custom protocol specification, we should leverage the strengths of the underlying SOAP specification. This would keep the HPCP specification simpler and easier to implement. Many thousands of hours have gone into the standards process for SOAP, SAML, XML Signatures and the other standards we wish to utilize. Rather than implement these standards in our own proprietary way, we need to join with the rest of the computer industry and embrace the spirit of Web Services as well as the standards defined today.

Appendix A - WSDL for DTSDataTypes

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="DataTransportStandardDataTypes"
    targetNamespace="http://www.datatransportstandard.com/wsdl/D
    TSDataTypes.wsdl" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:coremain="http://schemas.pescxml.org/0002/xsd/Core"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <import location="../Schema/CoreMain.xsd"
      namespace="http://www.standardscouncil.org/docs/xml_tran
      script/CoreMain.xsd" />
  - <types>
    - <xsd:schema
        targetNamespace="http://www.datatransportstandard.com
        /wsdl/DTSDataTypes.wsdl"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:coremain="http://schemas.pescxml.org/0002/xsd/
        Core"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <xsd:simpleType name="TransmitDateTimeGMT">
        - <xsd:annotation>
            <xsd:documentation>Required. This field is the
              DateTime in GMT. The value for this field
              must be set as the last part of the
              transmission process, immediately prior to
              transport.</xsd:documentation>
          </xsd:annotation>
          <xsd:restriction base="xsd:dateTime" />
        </xsd:simpleType>
      - <xsd:simpleType name="TransportUUID">
        - <xsd:annotation>
            <xsd:documentation>Required. Based on the
              W3C UUID definition.</xsd:documentation>
          </xsd:annotation>
          <xsd:restriction base="xsd:string" />
        </xsd:simpleType>
      - <xsd:element name="Sender">
        - <xsd:complexType>
          - <xsd:sequence>
              <xsd:element maxOccurs="1" minOccurs="1"
                name="sender"
                type="coremain:EntityType" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      - <xsd:element name="Recipient">
        - <xsd:complexType>
          - <xsd:sequence>
```

```
                <xsd:element maxOccurs="1" minOccurs="1"
                    name="recipient"
                    type="coremain:EntityType" />
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:schema>
    </types>
</definitions>
```

Appendix B – WSDL for getStatus

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="DataTransportStandard"
    targetNamespace="http://www.datatransportstandard.com/wsdl/getStatus.wsdl" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:dts="http://www.datatransportstandard.com/wsdl/DTSDataTypes.wsdl"
    xmlns:stat="http://www.datatransportstandard.com/wsdl/getStatus.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <import location="../Schema/CoreMain.xsd"
      namespace="http://schemas.pescxml.org/0002/xsd/Core" />
    <import location="DTSDataTypes.wsdl"
      namespace="http://www.datatransportstandard.com/wsdl/DTSDataTypes.wsdl" />
  - <message name="getStatusRequest">
      <part name="payload" type="xsd:string" />
      <part name="transportUUID" type="dts:TransportUUID" />
      <part name="transmitDateTimeGMT"
        type="dts:TransmitDateTimeGMT" />
      <part name="sender" element="dts:Sender" />
      <part name="recipient" element="dts:Recipient" />
    </message>
  - <message name="getStatusResponse">
      <part name="payload" type="xsd:string" />
      <part name="transportUUID" type="dts:TransportUUID" />
      <part name="transmitDateTimeGMT"
        type="dts:TransmitDateTimeGMT" />
      <part name="sender" element="dts:Sender" />
      <part name="recipient" element="dts:Recipient" />
    </message>
  - <portType name="DataTransportStandardPortType">
    - <operation name="getStatus">
        <documentation>Required Operation. Response codes are
          based on RFC 2616.</documentation>
        <input message="stat:getStatusRequest" />
        <output message="stat:getStatusResponse" />
      </operation>
    </portType>
  - <binding name="DataTransportStandardBinding"
      type="stat:DataTransportStandardPortType">
      <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    - <operation name="getStatus">
        <soap:operation soapAction="urn:dts:getStatus"
          style="document" />
      - <input>
```

```
            <soap:body
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" parts="payload" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="recipient" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="sender" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="transportUUID" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="transmitDateTimeGMT" use="encoded" />
          </input>
        - <output>
            <soap:body
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" parts="payload" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="recipient" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="sender" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="transportUUID" use="encoded" />
            <soap:header
              encodingStyle="http://schemas.xmlsoap.org/soap
              /encoding/" message="stat:getStatusRequest"
              part="transmitDateTimeGMT" use="encoded" />
          </output>
        </operation>
      </binding>
    </definitions>
```

Appendix C – Endpoint

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="urn:getStatus"
    targetNamespace="http://www.nchelp.org/DataTransportStandard.
    wsdl" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:stat="http://www.datatransportstandard.com/wsdl/getStat
    us.wsdl">
    <import location="getStatus.wsdl"
      namespace="http://www.datatransportstandard.com/wsdl/ge
      tStatus.wsdl" />
  - <service name="DataTransportStandard">
    - <port binding="stat:DataTransportStandardBinding"
        name="DataTransportStandardPort">
        <soap:address
          location="http://localhost:8000/ccx/DataTransportSt
          andard" />
      </port>
    </service>
  </definitions>
```