Network Node Functional Specification

Version 1.1

September 17, 2003

Task Order No.: T0002AJM038

Contract No.: GS00T99ALD0203

Abstract

This Functional Specification provides a detailed description of an Exchange Network Node's expected behavior including function invocation and expected output.

Amendment Record

Version	Date	Amended By	Nature of Change
Version 1.1	September 17, 2003	A. Reisser	The return type of Query was changed from queryResults to xsd:string. Clarified the parameters of positioned fetch (rowId and maxRows). rowId must be 0, and maxRows must be -1 if positioned fetch is not requested. Added Solicit as a ServiceType in the GetServices method. This allows user to retrieve a list of service requests supported by the Solicit method. Clarified the transactionId parameter in the Download parameter, the parameter may be empty for pre-established or ad hoc download operations.

Table of Contents

Table of Tables

Table of Figures

Foreword

The Network Exchange Protocol V1.1 (Protocol) and the Network Node Functional Specification V1.1 (Specification) define the conversation between and the behavior of Nodes on the Environmental Information Exchange Network (Exchange Network). The Network Steering Board (NSB) expects the Protocol and Specification to have a shelf life of between 12-24 months. As a result, the documents are forward-looking. They define and describe certain functionalities that will not immediately be utilized but are expected to become paramount as the Exchange Network evolves during its initial implementation. For example, the documents discuss and describe UDDI and other Registries as integral parts of the Network. Their use is implicit in the Protocol and Specification, but currently no official registries exist but they do merit discussion in these documents as it is expected that they will exist in the next 12-24 months.

These documents, in their first generation, were/are designed to support relatively simple state and EPA dataflows. They do so by proposing a small number of primitive Network Web services which Network Partners group into larger (but still simple) transactions to flow data. Most of these transactions are now conducted manually through the use of terminal/host clients, email, ftp, http uploads or diskettes. These Web services are:

- Authenticate
- NodePing
- GetServices
- GetStatus
- Notify
- Download
- Submit
- Solicit
- Query
- Execute (Optional method. Refer to Paragraph 8.1)

As indicated by the “Authenticate” service, the Protocol and Specification present a decentralized approach for authentication. Each Network Partner is responsible for authenticating users of their Nodes. While allowing optimum flexibility and ultimate control of authentication at the level of the Network Partner, decentralizing authentication could place a resource burden on future Network Partners. The USEPA as part of their Central Data Exchange (CDX) have created the Network Authorization and Authentication Service (NAAS). Any Network Partner can use this service to authenticate users. An additional Web service “Validate,” is required, to use the NAAS. The use of the NAAS is described in a separate document, the Network Security Guidelines and Recommendations V1.0 found on the Exchange Network Website. It is expected that in the next 12-24 months, authorization service will be made available at the NAAS. The “Authenticate” service (the process of determining the identity of a subject - not just limited to users; it could, and often should, apply to machines and messages in a secure environment) is nebulous with respect to Nodes or clients. That is, any Node or client can use the “Authenticate” service to obtain authentication. As a result, all potential data exchanges are supported.

As in any software project, these documents represent a series of design decisions and compromises. In their entirety, the Protocol and Specification will strike some implementers as

overly complex, and others (or maybe some of the same) as rudimentary. While these documents, created as part of a pilot project, went through several iterations, and represent the most current Network knowledge, the NSB acknowledges that these documents will need updates for several possible reasons including advances in technology.

Critical note to Node implementers:

A WSDL file accompanies the Protocol and Specification. The WSDL file is machine-readable and is the canonical description of the Protocol and Specification. Node implementers should use the WSDL file(s) as the starting point for their Node and client development. Each Node will have to customize the generic WSDL file for their Node. The ability to generate code from the WSDL file is an essential feature of most SOAP toolkits.

Acknowledgements

This document has been developed through the support and analytic contributions of a number of individuals and programs within EPA, ECOS, and several state agencies. These individuals offered valuable insight, lessons learned from their work on this and prior Node workgroups, and hard work in creating the Node V1.0 Specification and Protocol.

State Participants

Dennis Burling (State CoChair), Nebraska Department of Environmental Quality

David Blocher, Maine Department of Environmental Protection

Harry Boswell, Mississippi Department of Environmental Quality

Dan Burleigh, New Hampshire Department of Environmental Services

Frank Catanese, New Hampshire Department of Environmental Services

Ken Elliott, Utah Department of Environmental Quality

Dave Ellis, Maine Department of Environmental Protection

Renee Martinez, New Mexico Environment Department

Tom McMichael, New Mexico Environment Department

Melanie Morris, Mississippi Department of Environmental Quality

Dennis Murphy, Delaware Department of Natural Resources and Environmental Control

Brent Pathakis, Utah Department of Environmental Quality

Brian Shows, Mississippi Department of Environmental Quality

Chris Simmers, New Hampshire Department of Environmental Services

Michael Townshend, Delaware Department of Natural Resources and Environmental Control

Robert Williams, Maine Department of Environmental Protection

Karen Knox, Maine Department of Environmental Protection

EPA Participants

Connie Dwyer (EPA CoChair), Office of Environmental Information

Chris Clark, Office of Environmental Information

Patrick Garvey, EPA NSB Executive Staff

Environmental Council of States

Molly O'Neill, ECOS NSB Executive Staff

Support Contractors

Dave Becker, Computer Sciences Corporation

Tom Potter, Computer Sciences Corporation

Glenn Tamkin, Computer Sciences Corporation

Yunhao Zhang, Computer Sciences Corporation

Andrea Reisser, Concurrent Technologies Corporation

Kochukoshy Cheruvettolil, Ross & Associates Environmental Consulting, Ltd.

Louis Sweeny, Ross & Associates Environmental Consulting, Ltd.

Rob Willis, Ross & Associates Environmental Consulting, Ltd.

State Contractors/Consultants

Tony Pruitt, Ciber Federal Solutions

Steven Wu, enfoTech & Consulting Inc.

Chris McHenry, Integro

Calvin Lee, Oracle

Brad Loveland, Venturi Technology Partners

Brett Stein, XAware Inc.

1.0 Introduction and Terminology

1.1 Introduction

This document describes the expected behavior of a Network Node. It defines the functions the Node performs, how it invokes these functions, and the output expected.

1.2 Terminology

Term	Definition/Clarification
CID	Content ID
DBMS	Database Management System
DET	Data Exchange Template
DIME	Direct Internet Message Encapsulation
EPA	Environmental Protection Agency
Exchange Network	Environmental Information Exchange Network
NAAS	Network Authentication and Authorization Services. This is a set of centralized security services shared by all Network Nodes.
PKI	Public Key Infrastructure
RPC	Remote Procedure Calls
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TRG	Technical Resource Group
UML	Unified Modeling Language. The industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
URL	Uniform Resource Locator
UUID	Universal Unique Identifiers
W3C	World Wide Web Consortium
WSDL	Web Service Definition Language. An XML format for describing Network services as a set of endpoints operating on messages. Message definitions in WSDL are used in this document.
XML	Extensible Markup Language
XML Namespace	XML Namespace is a collection of names, identified by a URI reference. Namespaces in XML documents provide processing context and prevent name collisions

1.3 Principles, Assumptions, and Constraints

Principles are rules or maxims that guide subsequent decisions. Principles consist of a list of criteria involving business direction and good practice to help guide the architecture and design.

Assumptions are expectations that form the basis for decisions, which if proven false, would have a major impact on the project. They identify key characteristics of the future that are assumptions for the architecture and design, but are not constraints.

Constraints are restrictions that limit options. They are typically things that must or must not be done when designing the application. They identify key characteristics of the future that are accepted as constraints to architecture and design.

The principles, assumptions, and constraints for the Network Node Functional Specification V1.0 are:

1. The specification is expected to have a life of 18-24 months. During this time, actual Network usage information will be used to develop V2.0.
2. The specification will be kept as simple as possible. This is to ensure interoperability without unreasonable Network participation criteria.
3. Immediate development of the specification is required because:
   - Network participants need the specification to assist their Node implementations.
   - The Network Implementation Plan calls for ten (10) Nodes implemented by Q2 2003. However, a few dozen State agencies began establishing Nodes in 2002.
   - Even if the initial specification is imperfect and incomplete, the Network will work more efficiently and effectively with Network standardized expectations, functional performance standards, and “rules.”
   - Given the flexibility of Network technologies, implementers will be looking for all practical guidance available.
4. The specification must be consistent with the Network Exchange Protocol V1.0.
5. The specification must be consistent with the Network Security Guidelines provided in a separate document.
6. The specification must be consistent with the Network Registry Guidelines and operation.

1.4 Requirements

These requirements describe what will be delivered as part of the Network Node Functional Specification Version 1.0. The Network Node Functional Specification V1.0 shall:

1. Support all critical requirements for dataflows including the ability to “package” the relevant data using extensible markup language (XML) schemas developed by exchange partners and Network participants.
2. Use HTTP, Web Services Description Language (WSDL), and Simple Object Access Protocol (SOAP). Emerging industry standards will be used as consistently as possible in the application of these protocols.
3. Implement, and be compliant with, security procedures identified in the Network Exchange Protocol V1.0. If the Network Security Guidelines become available during the shelf life of the protocol, they will supercede security measures outlined herein.
4. Be implemented using the most common toolsets in use by Node implementers. A high degree of customization will be avoided.

2.0 Namespaces and Encoding Rules

Messages defined in this specification use either SOAP/Remote Procedure Call (RPC) encoding (also known as Section 5/Section 7 encoding) or Document/Literal encoding.

The SOAP encoding is governed by rules in SOAP Section 5 specifications, while messages in Document/Literal encoding must conform to the specified schema.

For purposes of the Network Node 1.0 project, the default XML namespace for data types and structures is:

http://www.exchangenetwork.net/schema/v1.0/node.xsd

The target namespace used by the corresponding WSDL file is:

http://www.exchangenetwork.net/schema/v1.0/node.wsdl

Versioning information is introduced into the schema from V0.9 and forward. Since the namespace URL is in all request and response messages, both service providers and requesters will be able to deal with different versions smoothly.

The namespaces without a version number are considered to be V0.8. For example, V0.8 was not supported after V0.9 was deployed to the initial Node 1.0 Group. Similarly, V0.9 will not be supported after V1.0 is deployed. V1.0 is the only normative specification. The actual deployment of the version level is a future enhancement to the system and that will likely be supported. The version levels currently being deployed are to create future versioning positioning.

The Technical Resource Group (TRG) Data Exchange Template (DET) workgroup is developing guidance on versioning for Network activities that will be incorporated upon completion and approval.

3.0 Data Elements and Structures

3.1 Document Types

The unit of exchange in the Network Exchange Protocol V1.0 is the document. Although documents can be in many different forms, they are classified into three (3) major categories:

1. Structured Document: Structured documents conform to a predefined structure. Documents, in document/literal encoding, carried in the SOAP message header or body are structured documents. External XML documents attached to SOAP messages are also structured.
2. Unstructured Document: Documents that do not have a predefined structure fall into this category. Examples include word documents, flat files, and binary files.
3. Relational Document: Relational documents are structured documents with relational constraints imposed on internal data elements. Records from a relational database are considered relational.

The Network Exchange Protocol V1.0 facilitates document exchanges of all three (3) categories. Table 1 shows how Network exchange interfaces provide support for these documents.

Document Type	Interface	Carrier	Comment
Structured	Send, Retrieve	Internal /Attachment	
Unstructured	Send, Retrieve	Attachment	
Relational	Database	Internal	Document embedded in message body

Table 1: Network Exchange Interface Support

3.2 Document Structure (nodeDocument)

A document in this protocol is defined using XML schema, as a complex data type (a structure):

```
<complexType name="nodeDocument">
        <sequence>
          <element name="name" type="xsd:string"/>
          <element name="type" type="xsd:string"/>
          <element name="content" type="xsd:base64Binary"/>
        </sequence>
</complexType>
```

Where **name** is the file name, **type** is one of the following:

- XML: An XML document.
- Flat: A flat text file.
- Bin: A binary file.
- ZIP: A compressed file in ZIP format.
- OTHER: An unspecified or unknown file type.

Note: this list of nodeDocument types will be expanded as needed to accommodate new types.

Note the sequence tag in the definition indicates that all children must be in a sequential order as specified. The value of the content element is the actual document, and base64 encoded if embedded in the structure. If the document is an attachment, the content element should be empty, but with an **href** attribute of content Id (CID) referencing the attached document. The following example shows a structure with an attached document:

<q3:myDoc xsi:type="q3:nodeDocument" xmlns:q3="http://www.exchangenetwork.net/xsd">

```
<name                    xsi:type="xsd:string">mydata.xml</name>
<type                          xsi:type="xsd:string">XML</type>
<content  xsi:type="xsd:string"  href=" f9647203-a4b9-4b1b-bd3c-
8186f75698bc"></content>
```

</q3:myDoc>

where **f9647203-a4b9-4b1b-bd3c-8186f75698bc** is a reference to the actual attachment outside of the SOAP message part.

3.3 SOAP Attachments

In a document exchange process, payloads can be any type of file, including XML files, text files, and binary files. It is recommended that the payloads be sent as attachments under the following conditions:

- The file is not a well-formed XML file.
- The document is large.

There are two (2) standards available for attachments: SOAP message with Attachment (SwA) and Direct Internet Message Encapsulation (DIME). Network Nodes must support DIME.

All attachments must be referenced in the SOAP main message body. Unreferenced attachments, which have no meaning to the receiver, will not be processed.

3.3.1 SOAP Message with DIME

DIME is a binary protocol originally proposed by Microsoft and IBM. The advantages of DIME are simplicity and performance. DIME attachments do not need to be encoded, which often produces a significant savings of time and resources. Each payload, including the main message body, is encapsulated in a DIME record. A DIME message is a set of records with the main SOAP message as the first record.

3.4 Fault Details

All fault messages **must** have a fault detail entry that contains error information specific to Node operations. The fault detail is a child element of the detail element defined by SOAP 1.1. It is defined as:

```
<complexType name="faultdetails">
      <sequence>
        <element name="errorcode" type="xsd:string"/>
        <element name="description" type="xsd:string"/>
      </sequence>
```

```
</complexType>
```

This is a simple structure with two (2) child elements: errorcode and description, both are type string. An example fault message with fault detail element is shown below.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
http://schemas.xmlsoap.org/soap/envelope/">

    <SOAP-ENV:Body> <SOAP-ENV:Fault>

           <faultcode>SOAP-ENV:client</faultcode>

           <faultstring>Invalid User</faultstring>

<detail><faultdetail
xmlns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

<errorcode>E_UnknownUser</errorcode>
<description>

Authentication failed; please check your userId and password.

</description>

</faultdetail>

             </detail>

</SOAP-ENV:Fault></SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

The message indicates that the failure is due to an invalid user name or password.

Note: The default namespace for fault detail is http://www.exchangenetwork.net/schema/v1.0/node.xsd*, representing a custom structure defined by this specification.*

The Network Exchange Protocol V1.0 list of predefined Exchange Network error codes is shown in Table 2.

Error Code	Description
E_UnknownUser	User authentication failed
E_Query	The supplied database logic failed
E_TransactionId	A transaction ID could not be found
E_UnknownMethod	The requested method is not supported
E_ServiceUnavailable	The requested service is unavailable
E_AccessDenied	The operation could not be performed due to lack of privilege
E_InvalidToken	The securityToken is invalid
E_TokenExpired	The securityToken has expired
E_FileNotFound	The requested file could not be located
E_ValidationFailed	DET validation error
E_ServerBusy	The service is too busy to handle the request at this time, please try later
E_RowIdOutofRange	The rowId parameter is out of range
E_FeatureUnsupported	The requested feature is not supported
E_VersionMismatch	The request is a different version of the protocol
E_InvalidFileName	The name element in the nodeDocument structure is invalid
E_InvalidFileType	The type element in the nodeDocument structure is invalid or not supported
E_InvalidDataFlow	The dataflow element in a request message is not supported
E_InvalidParameter	One of the input parameters is invalid
E_InternalError	An unrecoverable error occurred during processing the request
E_InvalidSQL	Syntax error in the SQL statement
E_AuthMethod	The authentication method is not supported
E_AccessRight	User privilege is insufficient for the operation

Table 2: Exchange Network Error Codes

In addition to the error codes listed above, service providers may return the native database management system (DBMS) error code if a database operation fails.

The description element in fault detail is a human readable string description of the error. It should contain as many details as possible so that the error can be avoided in subsequent requests.

4.0 Logging

All Network Nodes must log received transactions in a persistent storage area and provide search capability for tracking transactions either by transaction ID or requester's ID. In addition to information about submitted documents, the log record should contain the following information, at a minimum: requester's ID, time received, transaction status. Additional information may be provided.

It is also recommended that an activity log, a log that contains detailed processing steps, be provided to assist problem finding and debugging.

5.0 Network Service Interfaces

Network services defined in the Network Exchange Protocol V1.0 are classified into four (4) major abstract interfaces:

1. Send Interface: A group of methods for submitting documents and other basic Network services.
2. Database Interface: A set of methods for database operations.
3. Retrieve Interface: A set of methods for event notification and polling, and document retrieval.
4. Administration Interface: Methods for Network-wide coordination and management.

Implementation of all the interfaces is mandatory, although a Node may elect to support only limited database processing commands in Execute and Query. Web methods in each interface are listed in Table 3.

Interface	Methods
Send	Authenticate, Submit, GetStatus
Database	Query, Solicit, Execute
Retrieve	Notify, Download
Administration	NodePing, GetServices

Table 3: Methods Supported in Each Interface

Figure 1 shows a static Unified Modeling Language (UML) diagram of interfaces in a Network Node. Note that a Node can own more than one instance of each interface. The database interface, as well as the notification interface, uses a nodeDocument data structure. The diagram also shows that at least one (the 1…* notion). Send Interface must be implemented by a Node.

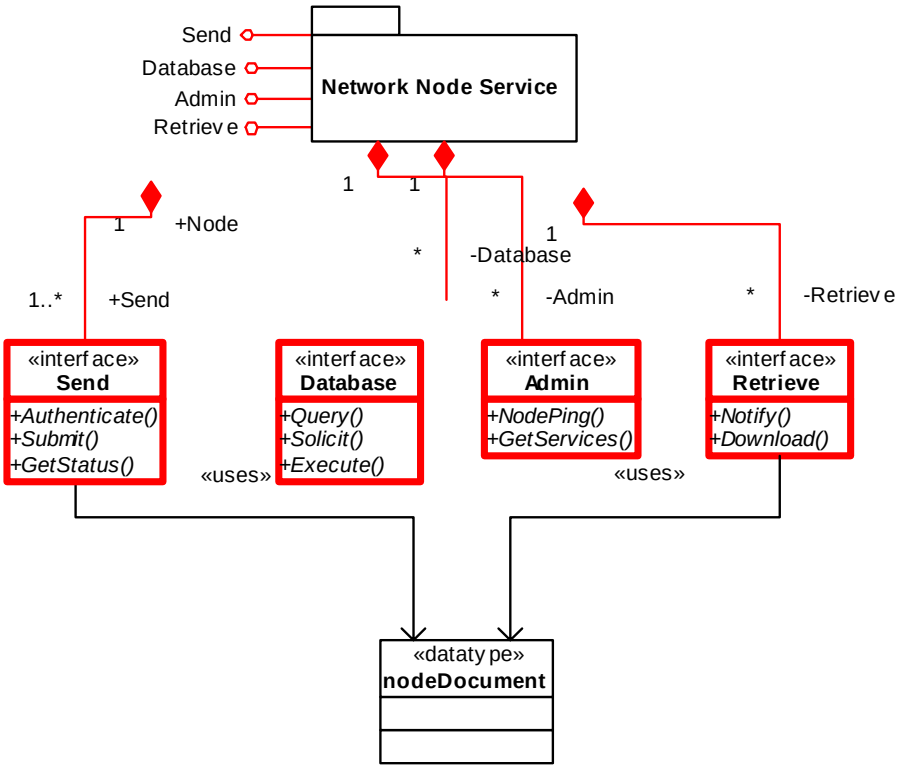

Figure 1. Static UML Diagram for Network Node Services

6.0 Node Web Methods

The Network Node Functional Specification V1.0 describes the behavior and interfaces of the service provider component. One of the design goals of this document is to create a framework of Web services such that data exchanges of any type between Nodes can be conducted seamlessly and automatically. The Web interface layer of the framework will create fully programmable environments on which clients can build automated tools, in any programming language, to send documents into the Network or to track previous submissions.

A Node is a service provider. Thus, the key interfaces that must be implemented in a Node include the following Web methods:

- Authenticate
- Submit
- Query
- GetStatus
- Notify
- Solicit
- Download
- NodePing
- GetServices
- Execute (Optional method. Refer to Paragraph 8.1)

This basic set of functions will be applicable for each given type of dataflow that will be exchanged through the Node, considering that each Node may be able to handle many kinds and types of data.

The following subsections define behaviors of each Web method, and give detailed descriptions of inbound/outbound messages.

6.1 Authenticate

6.1.1 Description

The Authenticate method authenticates a user using a supplied credential. It returns a securityToken when successful. The securityToken, also referred to as the securityToken, must be included in all other method invocations, except NodePing, as a proof of identity.

A securityToken is an opaque string that is meaningful only to the issuer or trusted peers. It may include, but is not limited to, the following information:

- The user ID or profile name.
- A session ID for state management.
- A timestamp for aging, expiration.

Service providers must implement an aging strategy to prevent replay attack. An expired token should be discarded immediately. A suggested token life span is about ten (10) minutes.

Authenticate messages must be sent through a secure transport such as secure socket layer (SSL). Note that although SSL is very good in securing communication channels, its usage, as an authentication system, is problematic; mutual verification of certificates in a large-scale distributed system is proven to be very expensive (public key infrastructure [PKI] required) and

difficult to implement. The securityToken scheme presented here offers a simple yet effective way of identification and authentication.

Note also that the specification itself does not define exactly how users are authenticated. Each Node implementer is free to choose any available authentication process in the underlying operating system. However, due to the Network connectivity, a security breach at one Node may have a grave impact to the overall operation. It is the responsibility of the Node operator to choose a secure authentication process. Network Security Guidelines and Recommendations, describing security practices for Network services, were provided in a separate document dated February 28, 2003.

As described in the accompanying Network Exchange Protocol V1.0 document delivered on March 14, 2003, and the Network Security Guidelines document, initial implementations will rely on an Environmental Protection Agency (EPA) hosted Network Authentication and Authorization Services (NAAS), supplemented as needed by local security services.

6.1.2 Definition

Authenticate messages are governed by WSDL message definitions below:

```
<message name='Authenticate'>
  <part name='userId' type='xsd:string'/>
  <part name='credential' type='xsd:string'/>
  <part name='authenticationMethod' type='xsd:string'/>
</message>
<message name='AuthenticateResponse'>
  <part name='return' type='xsd:string'/>
</message>
```

Where Authenticate is the request message; AuthResponse is the response. The definition indicates that the Authenticate request message consists of three (3) variables: userId, credential, and authenticationMethod all of type string. The response message contains a single string variable named ‘return’, which contains the securityToken.

6.1.3 Arguments

The Authenticate message requires three (3) parameters: userId, credential, and authenticationMethod. userId is the user ID of the person or system. The value of credential is the user’s credential for accessing the Network services.

The authenticationMethod parameter specifies which authentication methods are to be used. The default authenticationMethod, and the only method supported by the Network Node Functional Specification V1.0, is password. Possible future authentication methods may include, but are not limited to:

- Password: The credential parameter contains a clear password.
- Digest: The credential parameter contains a digest (sha1) of the user’s password.
- Certificate: The credential contains the user’s digital certificate.
- SAML: The credential contains an encoded SAML assertion.

6.1.4 Return

Upon successful authentication, the service provider returns a SOAP message with a securityToken that is placed in 'return'. The securityToken becomes a security ticket for all subsequent service requests.

The service provider returns a SOAP fault message under the following conditions:

- The user record is unknown.
- The supplied credential is incorrect.
- A server side fault/exception.

The SOAP fault message must contain a detail element with *E_UnknownUser* as the error code when authentication fails.

6.1.5 Example

A typical request message is:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:Authenticate
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
<userId xsi:type="xsd:string">JohnDoe</userId>
<credential xsi:type="xsd:string">T34ngPRN2345INt</credential>
<authenticationMethod xsi:type="xsd:string">password</authenticationMethod>
</mns:Authenticate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

and a positive response would be:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:AuthResponse
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
<return xsi:type="xsd:string">34BjT34ngPRN2345INt</return>
</mns:AuthResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

where **`34BjT34ngPRN2345INt`** is the securityToken. The securityToken, in its encrypted form, is meaningless to the holder, but contains crucial information to the issuer.

6.2 Submit

6.2.1 Description

The Submit method provides a generic way of sending one or more payloads to a service provider. Payloads other than the message body are encapsulated in an array of nodeDocuments. A payload can be embedded into a nodeDocument structure as a base64 encoded value, or as a separate attachment referenced by the nodeDocument.

A dataflow is a logical collection of certain kinds of documents, understandable to the sender and the ultimate receiver. Therefore, a dataflow can also be understood as a tag of the ultimate receiver of the payload. A dataflow can carry other information as well, such as Network events or asynchronous database requests. Such dataflows will be identified by special URLs. A Submit message can only target to one (1) dataflow at a time.

Network Nodes are required to process the SOAP main body of request messages, but are not required to understand the contents of attachments unless the Node is the target Node (ultimate receiver). For instance, a missing telephone number in a submitted document is not a SOAP error, but rather a process related error that should be dealt with differently.

6.2.2 Definition

```
<message name='Submit'>
  <part name='securityToken' type='xsd:string'/>
  <part name='transactionId' type='xsd:string'/>
  <part name='dataflow' type='xsd:string'/>
  <part name='documents' type='typens:ArrayofDoc'/>
</message>
<message name='SubmitResponse'>
  <part name='return' type='xsd:string'/>
</message>
```

6.2.3 Arguments

The Submit method accepts four (4) top-level arguments:

- securityToken: A security ticket issued by the service provider or a trusted service provider.
- transactionId: A transaction ID for the submission if the operation is a result of an asynchronous operation. It should be the transactionId associated with a previous solicited operation (See the Solicit method) if any. It should be empty if the Submit operation is independent.
- dataflow: The name of target dataflow.
- documents: An array of documents of type nodeDocument. Each nodeDocument structure describes a single attachment or payload.

6.2.4 Return

The Submit method returns, when successful, a transaction ID, which can be used to query status of the submission (see GetStatus method).

It returns a SOAP fault message with E_InvalidToken, E_AccessDenied or E_TokenExpired as the error code inside the fault detail element if the securityToken is invalid, insufficient or expired.

It returns a SOAP fault message (Client Fault) if one of the payloads in the message could not be processed.

6.2.5 Example

The following example shows a request message with two (2) referenced attachments. The payloads are targeted to a dataflow called TRI_ME.

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" …>

<SOAP-ENV:Body>

<mns:Submit xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

<securityToken type='xsd:string'>234tFaU1</securityToken>

<transactionId type='xsd:string'/>

<dataflow type='xsd:string'>TRI_ME</dataflow>

```
<documents soap-enc:arrayType="mns:nodeDocument[2]">
<item>
<name type='xsd:string'>My First Attachment.xml</name>
<type type='xsd:string'>xml</type>
<content href="f9647203-a4b9-4b1b-bd3c-8186f75698bc" type='xsd:base64Binary'/>
</item>
<item>
<name type='xsd:string'>My Second Attachment.txt</name>
<type type='xsd:string'>text</type>
<content href="fa0b4b41-15af-4bb5-8420-1353e3e66554" type='xsd:base64Binary'/>
</item>
  </documents>
</mns:Submit>
</s:Body>
</s:Envelope>
```

Note that `f9647203-a4b9-4b1b-bd3c-8186f75698bc` and `fa0b4b41-15af-4bb5-8420-1353e3e66554` are CIDs of the attachments. It is easy to retrieve an attached document using its CID. A document is an attachment if the *content* element has an href attribute, of either CID or universal unique identifier (UUID).

6.3 Query

6.3.1 Description

The Query method is a function in the Database interface. The method is intended to run a series of predefined information requests that return data in an XML instance document that

conforms to a predefined standard schema. Many predefined information requests will be standard across the Network and some maybe unique to a particular Node.

How the information requests are implemented is Node specific. Node implementers may choose different ways to implement the standard requests as long as the returned results conform to the XML schema.

For Network efficiency, the service provider is highly recommended, but not required, to support positioned-fetches where the requester can ask for a subset of the records within the overall result set. This feature is especially useful for interactive applications with graphical user interfaces where only a limited number of records can be displayed at a time.

Another case where positioned-fetch may be important is when the result set is so large that the Network connection between the requester and the provider will likely timeout. Positioned-fetch allows requesters to partition the whole result set into smaller chunks and thus avoid possible Network problems.

Unlike other methods, the response message of this method uses document / RPC encoding style because the format of result sets varies widely from query to query.

6.3.2 Definition

The Query messages are defined by the following WSDL segments:

```
<message name='Query'>
  <part name='securityToken' type='xsd:string'/>
  <part name='request' type='xsd:string'/>
  <part name='rowId' type='xsd:integer/>
  <part name='maxRows' type='xsd:integer/>
  <part name='parameters' type='typens:ArrayOfstring'/>
</message>
<message name='QueryResponse'>
  <part name='return' type=’xsd:string'/>
</message>
```

6.3.3 Arguments

The Query method requires the following arguments:

- securityToken: A security ticket issued by the service provider or a trusted security provider.
- request: The database query to be processed. It should be the name of a predefined information request.
- rowId: The starting row for the result set, it is a zero based index to the current result set. The value of rowId must be 0 if positioned-fetch is not requested.
- maxRow: The maximum number of rows to be returned. The service provider uses a default value if maxRow is 0 or negative. The value of maxRow must be -1 if positioned-fetch is not requested. A special value, -1, means all records from the current row (specified by rowId) to the end of the result set.
- parameters: An array of parameter values for the information request.

6.3.4 Return

The Query method returns a result set as string if successful. It must return a SOAP fault message when it fails. The fault detail element may contain an SQL error code and/or an error description from the native database system.

If the number of records returned is less than the value of maxRow, it means the end of the result set. The requester should stop subsequent-fetches.

The service provider must return a SOAP fault message (E_RowIdOutofRange) if the rowId is out of range of the whole result set. It must also return a SOAP fault (E FeatureUnsupported) if positioned-fetch is unsupported and rowId is greater than 0.

Note that an empty result set is not an error. The service provider must return a positive response with 0 records.

6.3.5 Example

Suppose exchange partners agree to honor a query request named GetFacByZipcode, which might correspond to the SQL statement in a stored procedure:

```
select * from FACILITY where zipcode = _zipcode
```

In which *_zipcode* is a parameter, the request message would be:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:Query xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
   <securityToken type='xsd:string'>32yFw3</securityToken>
   <request type='xsd:string'>GetFacByZipcode</request>
   <rowId type='xsd:integer'>0</rowId>
   <maxRows type='xsd:integer'>200</maxRows>
   <parameters soap-enc:arrayType="mns:ArrayOfstring[1]">
      <item type='xsd:string'>20001</item>
   </parameters>
</mns:Query>
</s:Body>
</s:Envelope>
```

As can be seen, the parameters array now has one item in it, (i.e., the value of *_zipcode)*.

When there are multiple parameters, the position of a parameter in the parameters array is significant; it must match to that in the database query request.

6.4 GetStatus

6.4.1 Description

GetStatus is a method for transaction tracking. Once submitted, a transaction enters into different processing stages. The GetStatus method offers the client a way of querying the current state of the transaction.

For Nodes that do not support staged transactions, the status of a submission degrades to a Boolean value*: Failed* or *Completed*.

6.4.2 Definition

The GetStatus method has simple request and response messages defined below:

```
<message name='GetStatus'>
  <part name='securityToken' type='xsd:string'/>
  <part name='transactionId' type='xsd:string'/>
</message>
<message name='GetStatusResponse'>
  <part name='return' type='xsd:string'/>
</message>
```

6.4.3 Arguments

The GetStatus method requires two (2) mandatory parameters: securityToken and transactionId. transactionId is a transaction identification returned by the Submit, Solicit or Notify method.

6.4.4 Return

The GetStatus method returns a string description of the current status if the operation is successful. A list of common status strings is defined below:

- Received: A submission was received by the service but has not been processed.
- Pending: One or more documents are to be downloaded and processed by the service.
- Processed: The submission has been processed by the Node, but is waiting to be delivered to the target Node (the ultimate destination).
- Completed: The submission is complete and accepted by the target Node.
- Failed: The submission has failed. The requester should resubmit.

The method returns a SOAP Fault with an error code of E_TransactId if the transaction ID is invalid; it returns a SOAP Fault with an error code of E_InvalidToken or E_TokenExpired if the securityToken is invalid or has expired.

6.4.5 Example

A requester may send the following message:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:GetStatus
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
   <securityToken type='xsd:string'>32yFw3</securityToken>
   <transactionId         type='xsd:string'>         8aa828c3-53a0-41ae-9760-
7d9f54158090</transactionId>
```

```
</mns:GetStatus>
</s:Body>
</s:Envelope>
```

A positive response could be:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
… >

<SOAP-ENV:Body>

<tns:GetStatusResponse
xmlns:tns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

<return xsi:type="xsd:string">received</return>

</tns:GetStatusResponse>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

6.5 Notify

6.5.1 Description

The Notify method has three (3) intended uses: document notification, event notification, and status notification described as follows:

- Document notification: A Node or client notifies a service provider about availability of some documents (soliciting). The service provider can retrieve the documents anytime.
- Event notification: A Node sends, or possibly broadcasts, an event that is of interest to other parties. Event messages can be security alerts, shutdown notices, and other Network management notes.
- Status notification: A service provider sends a message to a requester to provide the current status of a submission or service request.

In document notification, locations of the documents are provided in the nodeDocument structure. The service provider will use the same structure to download the available documents, so it is very important for requesters to include sufficient information so that the documents can be easily located.

This specification does not define the semantics of events, as they are operation specific. Service providers are free to state the specific meaning of Network events. The event URI, however, must be unique and have a prefix of:

```
http://www.exchangenetwork.net/node/event
```

6.5.2 Definition

The request and response are defined by the following WSDL messages:

```
<message name='Notify'>
    <part name='securityToken' type='xsd:string'/>
    <part name='nodeAddress' type='xsd:string'/>
    <part name='dataflow' type='xsd:string'/>
    <part name='documents' type='typens:ArrayofDoc'/>
```

</message>

```
<message name='NotifyResponse'>
    <part name='return' type='xsd:string'/>
  </message>
```

6.5.3 Arguments

The request message has the following arguments:

- securityToken: A security ticket issued by the service provider or a trusted security provider.
- nodeAddress: For document notification, the parameter contains a Network Node address where the document can be downloaded. It should contain the initiator's Node address, or be empty if not applicable, for event and status notifications.
- dataflow: The target dataflow that identifies an event or status if the value is http://www.exchangenetwork.net/node/event or http://www.exchangenetwork.net/node/status documents: An array of related documents.

Documents have different meanings depending on the value of the dataflow. Table 4 shows the relationships given different streams:

	nodeDocument		
Dataflow	**Name**	**Type**	**Content**
http://www.exchangenetwork.net/node/event	Name of the event	Type of the event	Description of the event
http://www.exchangenetwork.net/node/status	Transaction ID	Status String	A description of the status, or error message if the transaction failed.
Other	Name of the document	Type of the document	

Table 4: Dataflow and Document Relationship

6.5.4 Return

The returned value, if processed successfully, is a transaction ID for document notification. The ID can be used to query status of the submission (see GetStatus method). The returned value can be any other string signaling acceptance of the event or status in other cases.

6.5.5 Examples

The example below shows a document notification. The client, in this case, made available two (2) files: `http://example.com/myFile.xml` and `http://example.com/MyText.txt` for the service provider to retrieve later.

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
```

```
<SOAP-ENV:Body>
<mns:Notify xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
   <securityToken type='xsd:string'>3F4T322V</securityToken>
   <dataflow type='xsd:string'>TRI_ME</dataflow>
   <documents                        soap-enc:arrayType="mns:nodeDocument[2]">
      <item>
<name type='xsd:string'>http://example.com/myFile.xml</name>
<type type='xsd:string'>XML</type>
<content href="fa0b4b41-15af-4bb5-8420-1353e3e66554" type='xsd:base64Binary'/>
</item>
<item>
<name type='xsd:string'>http://example.com/MyText.txt</name>
<type type='xsd:string'>Flat</type>
<content              href="             fa0b4b41-15af-4bb5-8420-1353e3e66555"
type='xsd:base64Binary'/>
</item>
       </documents>
</mns:Notify>
</s:Body>
</s:Envelope>
```

The following example shows an event message, perhaps to announce the unavailability of a Network Node:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:Notify xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
   <securityToken type='xsd:string'>3F4T322V</securityToken>
   <dataflow
type='xsd:string'>http://www.exchangenetwork.net/node/event</dataflow>
   <documents                        soap-enc:arrayType="mns:nodeDocument[1]">
      <item>
<name type='xsd:string'>State Node 5</name>
<type type='xsd:string'>Down</type>
<content type='xsd:base64Binary'>The node will be down at 12:32:00 07/12/2002
and will not be available until 13:30:00 07/12/2002.</content>
   </item>
     </documents>
</mns:Notify>
</s:Body>
</s:Envelope>
```

Note that message in the content element is not base64 encoded for clarity. A status notification message is similar to:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>

<SOAP-ENV:Body>

<mns:Notify xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

   <securityToken type='xsd:string'>3F4T322V</securityToken>

   <dataflow
type='xsd:string'>http://www.exchangenetwork.net/node/status</dataflow>

   <documents                          soap-enc:arrayType="mns:nodeDocument[1]">
      <item>

<name type='xsd:string'> 8aa828c3-53a0-41ae-9760-7d9f54158090</name>

<type type='xsd:string'></type>

<content type='xsd:base64Binary'>Accepted</content>

</item>

     </documents>
</mns:Notify>
</s:Body>
</s:Envelope>
```

It indicates that a transaction with ID `8aa828c3-53a0-41ae-9760-7d9f54158090` has been accepted.

6.6 Solicit

6.6.1 Description

The Solicit method performs the requested operation in the background or sometimes offline. It is designed especially for queries that may take a long time.

In most situations, the Solicit method is used to ask for a Query operation. The service provider may kick off the Query operation immediately when the request is received, thus avoiding management of a transaction queue.

The service provider decides whether to process the transaction immediately or later. It may spawn a separate thread to process the request in a relatively low priority mode, or save the request in a transaction queue, that will be processed sequentially sometime later. However, it must return a transaction ID immediately to the requester, thereby acknowledging the acceptance of the transaction.

The service provider must return a SOAP fault message if the requested operation could not be honored.

Once the requested operation is processed successfully, the service provider should update the status of the transaction to *Complete*. If the operation failed for some reasons, the status of the transaction should be set to *Failed*.

The requester may optionally ask the service provider to submit the result to a Network Node location by specifying a returnURL. The location must contain a Network Node address that

has an implementation of the Submit method. It is the requester's responsibility to download the result if the returnURL is empty.

6.6.2 Definition

The Solicit messages are defined by the following WSDL segments:

```
<message name='Solicit'>
  <part name='securityToken' type='xsd:string'/>
  <part name='returnURL' type='xsd:string'/>
  <part name='request' type='xsd:string'/>
  <part name='parameters' type='typens:ArrayOfstring'/>
</message>
<message name='SolicitResponse'>
  <part name='return' type='xsd:string'/>
</message>
```

6.6.3 Arguments

The Solicit method requires the following arguments:

- securityToken: A security ticket issued by the service provider or a trusted security provider.
- returnURL: A Node address where results can be submitted. The service provider must call the Submit method at the specified address if it is not empty. If returnURL is empty, then it is the requester's responsibility to download the result.
- request: The operation to be performed. It is usually the name of a predefined information request.
- parameters: An array of parameter values for the information request.

6.6.4 Return

The method returns a transaction ID that can be used to check the status of the transaction.

6.6.5 Example

The following is a Solicit request for facility list within zip code 20001:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:Solicit xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
   <securityToken type='xsd:string'>32yFw3</securityToken>
   <returnURL type='xsd:string'></returnURL>
   <request type='xsd:string'>GetFacByZipcode</request>
   <parameters soap-enc:arrayType="mns:ArrayOfstring[1]">
      <item type='xsd:string'>20001</item>
   </parameters>
```

```
</mns:Solicit>
</s:Body>
</s:Envelope>
```

The requester will download the document when available.

6.7 Download

6.7.1 Description

The Download method is a function in the Retrieve Interface. It is different from the Submit method in two (2) ways:

1. Document flows from callee to caller (document retrieval).
2. The Download operation is usually initiated by a service provider.

The Download method is often used to fulfill a requested operation. For instance, after being notified by a submitter, a Node invokes the Download method to retrieve available documents.

If there is a pre-established contract between the two parties, e.g., names of the documents and their availability are predetermined, then a Node can actively retrieve the documents at fixed time periods without prior notification. The transactionId parameter may be empty in such cases.

The ability to download documents makes mutual data exchange possible. Any Node in the Exchange Network can be a service provider and, at the same time, a service consumer. From the caller's point of view, submitting is an operation of sending documents to a remote Node, while downloading is an operation of receiving documents from a remote Node.

Unlike the Submit method, however, the Download method gives access to some documents to the requester. The directory where the document resides should be limited to only those who have access rights. It is recommended that each user have a separate folder so that a document for one user cannot be accessed by another user.

6.7.2 Definition

The request message:

```
<message name='Download'>
    <part name='securityToken' type='xsd:string'/>
    <part name='transactionId' type='xsd:string'/>
    <part name='dataflow' type='xsd:string'/>
    <part name='documents' type='typens:ArrayofDoc'/>
  </message>
```

The response message:

```
  <message name='DownloadResponse'>
    <part name='documents' type='typens:ArrayofDoc'/>
  </message>
```

6.7.3 Arguments

The Download method takes the following parameters:

- securityToken: A security ticket issued by the service provider or a trusted security provider.
- transactionId: A transaction ID for the submission. It should be the same transaction ID issued by the Node (See the Notify method.) The parameter may be empty for a pre-established or ad hoc download.
- Documents: An array of nodeDocument structures. It should contain the same set of documents given by the Notify method.

When a Node is actively retrieving documents without prior notification, transactionId may contain a unique ID for the document to be retrieved. The documents parameter, in such a scenario, can be empty as long as the two parties know what documents are to be exchanged.

6.7.4 Return

The response message contains a dataflow identifier and a set of documents. Documents transmitted can be either embedded payloads or separate attachments.

6.7.5 Examples

The sample message below shows a Download request with two (2) documents:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>

<SOAP-ENV:Body>

<mns:Download
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

   <securityToken type='xsd:string'>3F4T322V</securityToken>

   <transactionId type='xsd:string'>2345-345</transactionId>

   <documents                        soap-enc:arrayType="mns:nodeDocument[2]">
           <item>

<name type='xsd:string'>http://example.com/myFile.xml</name>

<type type='xsd:string'>XML</type>

<content type='xsd:base64Binary'/>

</item>

<item>

<name type='xsd:string'>http://example.com/MyText.txt</name>

<type type='xsd:string'>Flat</type>

<content type='xsd:base64Binary'/>

      </item>
       </documents>
</mns:Download>
</s:Body>
</s:Envelope>
```

6.8 NodePing

6.8.1 Description

The NodePing method is a function in the Admin interface. It is a utility method for determining whether a Node is accessible. A positive response from the Node indicates that it is live and well. A Network error (no response) or SOAP Fault (not ready) means that the service is not available at this time.

NodePing is the only operation that does not require authentication.

6.8.2 Definition

```
<message name='NodePing'>
    <part name='Hello' type='xsd:string'/>
  </message>
  <message name='PingResponse'>
    <part name='return' type='xsd:string'/>
  </message>
```

6.8.3 Arguments

The NodePing method has one argument that may contain arbitrary text, preferably short or even null.

6.8.4 Return

The NodePing method returns a positive response in normal operational mode. It may return a SOAP fault if the service is not ready. The service provider should return the Table 5 service status codes in the positive response message.

Status	Meaning
Ready	The service is up and ready.
Busy	The service is heavily loaded, please call back later.
Unavailable	The service is currently unavailable.

Table 5: Service Status Codes

A Node can return other status codes in human readable form when needed. A response message without a status code is understood as **Ready**.

6.8.5 Examples

A NodePing example:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:NodePing
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
```

```
          <Hello />
      </mns:NodePing>
</s:Body>
</s:Envelope>
```

A positive response from the Node may be:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:NodePingResponse
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
          <return type='xsd:string' >Ready</return>
      </mns:NodePingResponse>
</s:Body>
</s:Envelope>
```

6.9 GetServices

6.9.1 Description

The GetServices method is a function in the Admin interface. It allows requesters to query services provided by a Network Node. The type of services that can be queried includes, but is not limited to:

- Interfaces: The Web service interfaces supported by the Node.
- Query: Predefined information requests that can be used in the Query method.
- Solicit: Predefined information requests that can be used in the Solicit method.
- Execute: Predefined procedures that can be used in the Execute method.

A Node may choose to support additional types (meta-data) when needed. To get a complete list of all service types, a requester can pass ServiceType as the value of the ServiceType element.

Using GetServices, a requester can determine the capability of a Node at runtime and proceed accordingly. On the other hand, it allows the service provider to extend the services provided, (e.g., add a new database report), without changing the infrastructure. The smart invocation and easy extensibility can greatly enhance the overall usability, stability and capability of the Exchange Network. See the Network Exchange Protocol V1.0 document for further discussion.

6.9.2 Definition

```
<message name='GetServices'>
    <part name='securityToken' type='xsd:string'/>
    <part name='ServiceType' type='xsd:string'/>
  </message>
  <message name='GetServicesResponse'>
    <part name='return' type='typens:Arrayofstrings'/>
  </message>
```

6.9.3 Arguments

The method requires a ServiceType string, which is defined as follows:

- ServiceType: A complete list of all service types that can be used as the value of the element.
- Interfaces: The Web service interfaces supported by the Node.
- Query: Predefined information requests supported by the Node.
- Execute: A list of predefined information requests provided by the Node.

Since service providers may elect to provide additional services, the method provides a capability of querying all ServiceTypes. A list of returned service types can then be used to get a list of services under a given service type.

6.9.4 Return

The returned message contains an array of all services of specified service type. An empty array should be returned if the service type is unknown or not supported.

If the ServiceType is Query or Execute, the Node must return a list of all predefined information requests suitable for being used as the argument for the Query and Execute methods. It must return an empty array with 0 items, not fault, if no query or procedure is provided.

6.9.5 Examples

The request message below gets a list of all service types from a service provider:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>

<SOAP-ENV:Body>

<mns:GetServices
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

      <securityToken type='xsd:string'>3F4T322V</securityToken>

          <ServiceType type='xsd:string'>ServiceType</ServiceType>

      </mns:GetServices>
</s:Body>
</s:Envelope>
```

The response message could be:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>

<mns:GetServicesResponse
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

<return xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[3]">

<Item xsi:type="xsd:string">Interfaces</Item>

<Item xsi:type="xsd:string">Query</Item>

<Item xsi:type="xsd:string">Execute</Item>

</return>

</mns:GetServicesResponse>
```

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This response message indicates that the service provider supports Query, among other things. The requester can call the method again, using Query as the value of ServiceType this time, to obtain a list of available information requests to be used as the parameter to the Query method. The following example demonstrates this:

The requester sends,

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:GetServices
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
      <securityToken type='xsd:string'>3F4T322V</securityToken>
          <ServiceType type='xsd:string'>Query</ServiceType>
      </mns:GetServices>
</s:Body>
</s:Envelope>
```

The provider may send a response message:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<mns:GetServicesResponse
xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
<return xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[4]">
<Item xsi:type="xsd:string">GetMyTransaction</Item>
<Item xsi:type="xsd:string">GetFacilityByName</Item>
<Item xsi:type="xsd:string">GetFacilityById</Item>
<Item xsi:type="xsd:string">GetFacilityByChangeDate</Item>
</return>
</mns:GetServicesResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

which indicates that the Node support four (4) predefined information requests. XML schema definitions of the information requests may be provided in a separated XML document.

7.0 Node Validation

Node validation is a process to assure full compliance with this specification. It is conducted using a series of test messages. Test messages are request messages for verifying Node operations and validating responses.

Data exchanged through test messages can be discarded following validation. The service provider, however, must perform operations as requested.

Tests are conducted using the same set of methods defined in this specification, with special identifiers indicating that the messages are only tests. Table 6 shows the specification 1.0 test identifiers for all Web methods.

Web Method	Test Identifier	Comment
Authenticate	None	Authentication is required for all tests.
Submit	dataflow = test	
GetStatus	None	
Notify	dataflow = test	
Download	dataflow = test	A Node can choose a fixed number of fixes to return regardless of the file names in the message.
NodePing	None	
GetServices	None	

Table 6: Test Message Definitions

For example, if a dataflow name is "**test**" in a Submit message, then it is a validation message. The message may contain random, generated data, which can be discarded immediately. However, the service provider must return a valid transaction ID for status tracking.

For those methods that do not have any side effects, (e.g., read-only methods such as NodePing and GetServices), no test identifier is defined.

All Nodes participating in the Exchange Network can perform tests against other Nodes, and publish test results. Tests can be either positive (e.g., submitting correct documents) or negative (e.g., login with an incorrect ID). The overall test score of a Node is calculated as the total number of successful tests minus the number of self-tests divided by the number of testers. Unimplemented methods are counted as failures.

Online tools will be provided to automate testing of Network Nodes in accordance with the Network Node 1.0 WSDL.

8.0 Appendix

8.1 Execute

Due to complexity and extra security requirements, Execute is defined as an optional method.

8.1.1 Description

The Execute method is a function in the Database interface. It is used to run stored procedures or other predefined operations. There are two basic usage scenarios:

1. The requested operation is a predefined name of service requests, as natural extensions to interfaces defined in the document.
2. The requested operation is a stored procedure defined by the service provider; the requester references it and supplies necessary parameters.

The first case allows the service provider to extend the functionality defined in the specification and to offer “value added” services without changing the programming interface.

Different from Query, where database records are read-only, the Execute method may allow requesters to modify database contents. A higher privilege should be required for such operations to mitigate the risk of data corruption.

8.1.2 Definition

The Execute message is defined by the following WSDL segments.

```
<message name='Execute'>
  <part name='securityToken' type='xsd:string'/>
  <part name='request' type='xsd:string'/>
  <part name='parameters' type='typens:ArrayOfstring'/>
</message>
<message name='ExecuteResponse'>
  <part name='return' type=’xsd:string’/>
</message>
```

8.1.3 Arguments

The Execute method accepts three (3) arguments:

- securityToken: An authentication ticket issued by the service provider or a trusted security provider.
- request: The database logic to be processed. It can be either the name of an operation or the name of a procedure.
- parameters: An array of parameter values for SQL statements or stored procedures.

8.1.4 Return

The Execute method returns the number of rows affected by the request or procedure if successful. It returns a fault message in all other cases. The fault detail element must contain native error information if the database operation fails, (i.e., syntax error or constraint violation).

8.1.5 Example

In the following example, the requester asks the service provider to execute a stored procedure named PROC5:

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>
<SOAP-ENV:Body>
<mns:Execute xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
   <securityToken type='xsd:string'>32yFw3</securityToken>
   <request type='xsd:string'>PROC5</request>
   <parameters soap-enc:arrayType="mns:ArrayOfstring[0]"/>
</mns:Execute>
</s:Body>
</s:Envelope>
```

9.0 References

1. Network Exchange Protocol V1.0, a deliverable to the EPA by CSC, March 14, 2003.
2. Advanced SOAP for Web Development, Dan Livingston, Copyright 2002, Prentice Hall PTR, Upper Saddle River, NJ, 07458.
3. Web Services Essentials, Ethan Cerami, Copyright 2002, O'Reilly & Associates, Sebastopol, CA, 95472.
4. Programming Web Services with SOAP, James Snell, Doug Tidwell, Paul Kulchenko, Copyright 2002, O'Reilly & Associates, Sebastopol, CA, 95472.
5. WS-Security, version 1.0. April 5, 2002.
6. W3C Node "Simple Object Access Protocol (SOAP) 1.1", May 22, 2000. (See http://www.w3.org/TR/2000/NOTE-SOAP-20000508/).
7. W3C Working Draft "SOAP Version 1.2 Part 1: Messaging Framework", Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 26 June 2002 (See http://www.w3.org/TR/2002/WD-soap12-part1-20020626.)
8. W3C Working Draft "SOAP Version 1.2 Part 2: Adjuncts", Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 26 June 2002 (See http://www.w3.org/TR/2002/WD-soap12-part2-20020626.)
9. W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See http://www.w3.org/TR/1999/REC-xml-names-19990114/.)
10. W3C Node "SOAP Messages with Attachments", John J. Barton, Satish Thatte, Henrik Frystyk Nielsen, December 11, 2000.
11. Internet Draft " Direct Internet Message Encapsulation (DIME)", Henrik Frystyk Nielsen, Henry Sanders, Russell Butek, Simon Nash, June 17, 2002.
12. W3C Node "Web Services Description Language (WSDL) 1.1", Erik Christensen, Francisco Curbera,Greg Meredith, Sanjiva Weerawarana, March 15, 2001 (See http://www.w3.org/TR/wsdl).
13. UDDI Version 3 Specification - http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, July 19, 2002.