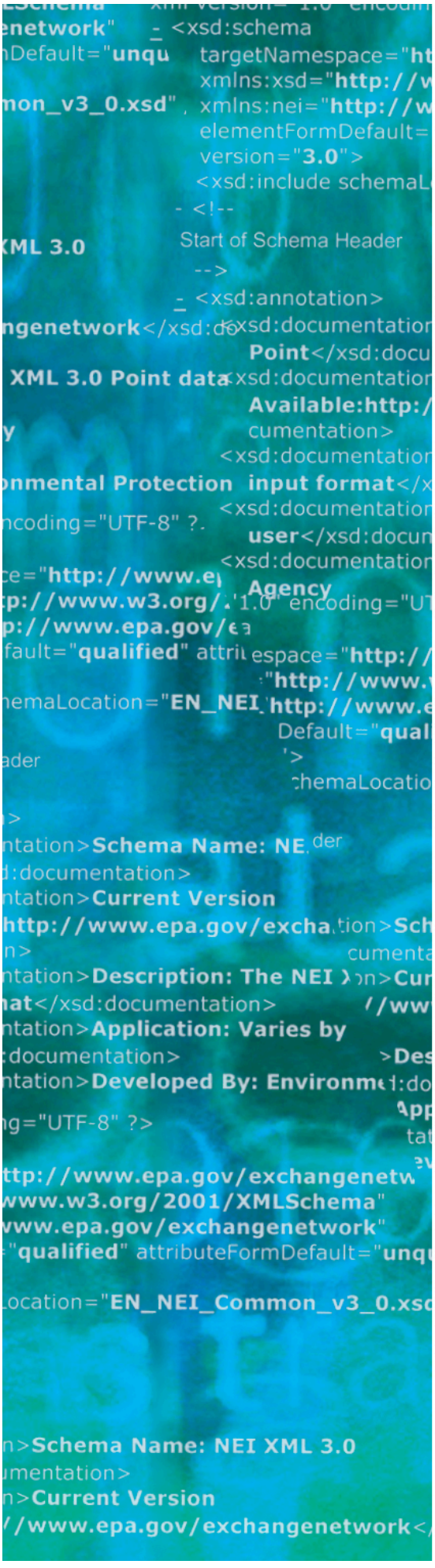

Network Node Functional Specification Version 2.1

Revised Date: June 24, 2011

Environmental Information eXchange

Abstract

This Functional Specification provides a detailed description of the expected behavior of an Exchange Network (EN) Node, including function invocation and expected output.

Revision History

Change Record			
Version Number	**Description of Change**	**Change Effective Date**	**Change Entered By**
2.0	Final 2.0 Specification	June 2, 2008	Dr. Yunhao Zhang
2.1	• Changed encoding style of node to all MTOM (Section 5) • Clarified transaction status code and descriptions (Section 3.5, 7.8) • Added more descriptive language for Fault error codes (Section 3.8) • Added a section on Node Interoperability Considerations (Section 6)	March 16, 2011	Dr. Yunhao Zhang

Table of Contents

Table of Tables

Table of Figures

1 Introduction and Terminology

1.1 Introduction

This document describes expected behavior of an Exchange Network Node version 2.1. It defines functions the node performs, how it invokes these functions, and the expected output.

1.2 Terminology

Term	Definition/Clarification
CID	Content Id
DBMS	Database Management System
DET	Data Exchange Template
DIME	Direct Internet Message Encapsulation
EN	Exchange Network
EPA	Environmental Protection Agency
MTOM	Message Transmission Optimization Mechanism
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

Term	Definition/Clarification
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 V2.1 are:

1. The specification will be kept as simple as possible. This is to ensure interoperability without unreasonable Network participation criteria.
2. The version 2.1 node specification will preserve sound features in version 1.0 specification so that the existing node components can be reused whenever possible.
3. The version 2.1 specification will not be compatible with version 1.0 because the SOAP and SOAP attachment protocols have changed. Therefore, version 1.0 nodes will be supported in parallel for a period of time to insure a smooth transition to the version 2.1 specification.
4. The specification must be consistent with the Network Exchange Protocol V2.1.
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 2.1. The Network Nodes implementing the Functional Specification version 2.1 shall:

1. Support all critical requirements for data flows including the ability to “package” the relevant data using Extensible Markup Language (XML) schemas developed by Exchange Network partners.

2. Support large payloads for data publishing.
3. Use SOAP 1.2 and MTOM (Message Transmission Optimization Mechanism) for all request and response messages. Emerging industry standards will be used as consistently as possible in the application of these protocols.
4. Implement, and be compliant with, security procedures identified in the Network Exchange Protocol version 2.1.
5. Have access to an SMTP server for Nodes implementing the NotificationURI and Recipient functionality.

2 Namespaces and Encoding Rules

Messages defined in this specification use only Document/Literal encoding.

For purposes of the Network Node 2.1 project, the default XML namespace for data types and structures is:

http://www.exchangenetwork.net/schema/node/2

The target namespace used by the corresponding WSDL file is:

http://www.exchangenetwork.net/wsdl/node/2

Throughout this document, **typens** is used as the prefix for the data type namespace and **tns** (target namespace) is used as the prefix for the WSDL definition namespace.

3 Data Types and Usage Conventions

3.1 Generic XML Type

In many data exchange scenarios, an Exchange Network Node needs to handle arbitrary XML documents where the schema is known, unknown, or yet to be defined. Such XML documents are defined as being GenericXMLType as below:

```
      <complexType name="GenericXmlType" mixed="true">
        <sequence>
          <any namespace="##any" minOccurs="1" maxOccurs="1"
          processContents="lax"/>
        </sequence>
        <attribute name="format" type="typens:DocumentFormatType"
use="optional" default="XML"/>
      </complexType>
```

The GenericXmlType is defined as a mixed-content type, allowing either a string value or an XML element as its child. The optional format attribute is used to indicate the content format for non-XML values. The content is assumed to be XML if the format attribute is missing.

The main purpose of defining GenericXmlType as mixed content is to support XML compression. The child XML element may be zipped and base64 encoded as a string value inside the GenericXmlType with the DocumentFormatType set to ZIP. EN clients receiving documents with the DocumentFormatType set to ZIP must first decode and then unzip the value to get the actual data.

In order to maintain simplicity and interoperability among all Exchange Network partners, uncompressed raw XML and compressed XML (using the ZIP archiving technology) are the only two valid format types that may be used in GenericXMLType, in addition to uncompressed string.

For XML data, the GenericXMLType must contain one, and only one, child element, which has a namespace other than http://www.exchangenetwork.net/schema/node/2. The XML processor should validate the contents of the child if a schema is specified.

The GenericXmlType is used in the Query method for returning arbitrary XML results governed by XML schema definitions and the GetServices method whose schema is defined outside this document. It is also employed by the Execute method for returning XML responses.

3.2 Document Types

A document, the unit of exchange in the Network Exchange Protocol version 2.1, can be formatted many different ways. The Exchange Network relies on three (3) common document definitions.

1. **XML Documents**: The most commonly used document type on the Exchange Network. XML documents are structured using an external, predefined schema, and may be included directly in the body of a SOAP message or attached outside of the SOAP envelope via the MTOM/XOP attachment mechanism.
2. **Non XML Documents:** Data can be in a wide range of formats.
3. **Compressed Documents:** Documents that have been reduced in size using the ZIP compression algorithm. Compressed documents have no predefined structure, but may contain structured (XML) contents when decompressed.

The Network Exchange Protocol V2.1 facilitates document exchanges of all three (3) categories. Table 1 shows how Network Exchange Interfaces provide support for these documents.

Document Type	Method	Carrier	Comment
XML	All Methods	Internal/Attachment	
Non-XML	Submit, Download	Internal/Attachment	
Compressed	All Methods	Attachment	In this case, Query may return compressed XML only

Table 1: Network Exchange Interfaces Support

3.3 Node Document Type

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

```
<complexType name="NodeDocumentType">
  <sequence>
    <element name="documentName" type="xsd:string"/>
    <element name="documentFormat" type="typens:DocumentFormatType"/>
    <element name="documentContent" type="typens:AttachmentType"/>
  </sequence>
  <attribute name="documentId" type="xsd:ID" use="optional" />
</complexType>
```

Where **DocumentName** is the file name, **DocumentFormat** is one of the following:

- XML: An XML document.
- FLAT: A flat text file.
- BIN: A binary file.
- ZIP: A compressed file (usually large XML datasets) in ZIP format.
- ODF: Open Document Format.
- OTHER: An unspecified or unknown file type.

The value of the **DocumentContent** element is the actual document.

The **DocumentContent** element is of **AttachmentType**, which is an extended base64Binary type defined as:

```
<complexType name="AttachmentType">
  <simpleContent>
    <extension base="xsd:base64Binary">
      <attribute ref="xmime:contentType" use ="required"/>
    </extension>
  </simpleContent>
</complexType>
```

The type has the attribute **xmime:contentType**, which must be a standard MIME content type. Commonly used contentTypes in the Exchange Network are listed in Table 2 below.

File Type	Content-Type
XML	text/xml
ZIP	application/zip
Image	image/png
Text	text/plain

Table 2: Commonly Used File Content Types

The optional attribute, **documentId**, should be constructed from a UUID which uniquely identifies a document within each Node. The **documentId** is of type XML:ID which requires the **documentId** to begin with an underscore (_) in combination with a letter or number.

e.g. `<typens:document DocumentId="_f654c35c-f223-4787-a947-8787f532d3fe">`

Note that a NodeDocument can be considered a generic content holder which could contain any object with a name, a type, and content.

3.4 Result Set Type

When a database query is executed, a network node returns a ResultSetType, which contains the result set and paging information. The ResultSetType is defined as:

```
<complexType name="ResultSetType">
  <sequence>
    <element name="rowId" type="xsd:integer" />
    <element name="rowCount" type="xsd:integer"/>
    <element name="lastSet" type="xsd:boolean"/>
    <element name="results" type="typens:GenericXmlType"/>
  </sequence>
</complexType>
```

- **rowId**: This is an integer for the first record contained in the result set sent to the requestor. This value must not be less than -1 and must not be more than the total number of rows in the result set minus one (n – 1). The **rowId** of the first record of a complete result set is always 0. If a result set with no results is returned, the **rowId** must be set to 0.
- **rowCount**: This is the number of records returned in the result set. This value must be set to 0 if no records are returned.
- **lastSet**: This is a Boolean value indicating if the data is the last result set. A value of *true* indicates no more data is available given the current parameters, and *false* means that more data is available. If the service provider supports positioned fetches, a consumer can retrieve the next result set by calling the Query method with a new **rowId** equal to the last rowId of the last results set plus the value of **rowCount**.
- **results**: This is a generic XML container that may hold any XML document, either compressed or uncompressed.

For a result set that may span multiple tables, the **rowId** and **rowCount** should be the properties of the first record in the set. See section 5.4 for more information on the expected functionality of Query and positioned fetches.

3.5 Status Response Type

When a request is issued, a Network Node processes the request and returns status information to the requester. The StatusResponseType is an XML structure that defines what should be included in the response. It is specified as an XML element with 3 children:

```
<complexType name="StatusResponseType">
  <sequence>
    <element name="transactionId" type="xsd:string" />
    <element name="status" type="typens:TransactionStatusCode" />
    <element name="statusDetail" type="xsd:string" />
  </sequence>
</complexType>
```

- **transactionId**: A UUID that uniquely identifies the transaction across all network nodes.
- **status**: A transaction status code:
    - Received: The transaction has been received by the Node but has not yet been processed or scheduled for processing.
    - Processing: The transaction is currently being processed.
    - Pending: Processing of the documents has not begun, and is either scheduled to be processed at a later time or is awaiting approval. Approved: The submission has been approved or certified if it needs

approval. However, the documents have not been delivered to the receiver yet.

- Processed: The request/submission has been processed at the node. However, any payload associated with the transaction has yet to be delivered to the final recipient, usually a backend process.
- Completed: The transaction has completed, no further action will be taken on the request/submission.
- Failed: The transaction has failed. No further action will be taken on the request/submission. The requester should reinitiate the transaction after the problem is fixed.
- Canceled: The transaction has been canceled by the node administrator or an approver.
- Unknown: The status of the transaction cannot be determined at this time.

- **statusDetail**: A string describes the current status. This value should provide additional detail useful to the user as to the particular nature of the status code.

3.6 Parameter Type

One of major changes in the Exchange Network Node Specification version 2.1 method invocations is the replacement of position-based parameter binding with name-based binding, necessitated by the transition to a Document/Literal WSDL. In version 2.1, parameters for data services and other web services are defined as:

```
<complexType name="ParameterType">
  <simpleContent>
    <extension base="xsd:string">
      <attribute name="parametnerName" type="xsd:string" use ="required" />
      <attribute name="parameterType" type="xsd:QName" use="optional"/>
      <attribute name="parameterEncoding" type="typens:EncodingType"
use="optional" default="None"/>
    </extension>
  </simpleContent>
</complexType>
```

ParameterType is an extension of xsd:string with the following attributes:

- **Name**: The name of the parameter.
- **Type**: The simple XML schema type of the parameter. It must be a qualified name such as xsd:string. This attribute is optional and a parameter must be a string if Encoding is not specified or the value of **Type** is *None*.
- **Encoding**: The encoding type of the parameter. See Table 3 for enumerated encoding types. This attribute is optional. If not specified, the parameter is assumed to be an unencoded string.

Encoding Type	Description
Base64	base64Binary encoded content.
ZIP	A base64 encoded string representing ZIP compressed content.
Encrypt	Content encrypted using triple-des algorithm. This is used for sending sensitive parameter such as passwords or social security numbers.
Digest	The content is a base64 encoded hash value of the parameter.
XML	XML structured contents. i.e., the parameter is an XML string.
None	No encoding.

Table 3: Parameter Encoding Types

For Query and Solicit requests, the Type and Encoding attribute may be present, but to maintain interoperability across all Nodes, only the XML encoding type should be used. When the Type and Encoding attributes are missing, the ParameterType is reduced to a simple element with a Name attribute and a string value. Exchange Network Nodes must accept and process parameters that are un-encoded strings or structured XML.

The ParamterType definition offers a powerful mechanism of passing arbitrary parameters to a node, including virtually all data types, plus parameter compression and encryption. The Type and Encoding attributes provide sufficient information for service providers to do late parameter binding, which is needed by the Execute method.

3.6.1 Parameter Binding

With name-based parameter binding, the position of the parameters in the request message is irrelevant. It is the service provider's responsibility to associate each parameter with the right value given its name. For example, the following parameters:

```
<parameter Name='FacilityName'>Exxon</parameter>
<parameter Name='ZipCode'>20001</parameter>
```

are equivalent to:

```
<parameter Name='ZipCode'>20001</parameter>
<parameter Name='FacilityName'>Exxon</parameter>
```

in the name-based parameter binding.

3.6.2 Parameter Semantics

When an array of parameters is passed to Query or Solicit, parameters with different names are associated using AND logic when querying the backend Database. Parameters with the same name are associated using OR logic when constructing database queries. For instance, the following parameter set indicates a search for records with the facility name Exxon and a zip code of 20001 OR 20006.

```
<parameter Name='FacilityName'>Exxon</parameter>
<parameter Name='ZipCode'>20001</parameter>
<parameter Name='ZipCode'>20006</parameter>
```

3.7 Notification Message Type

Notification message is an XML structure that allows a node to receive external events. It can be used for document notifications, transaction notifications and event notifications.

The NotificationMessageType is defined as follows:

```
      <complexType name="NotificationMessageType">
        <sequence>
          <element name="messageCategory"
type="typens:NotificationMessageCategoryType"/>
          <element name="messageName" type="xsd:string"/>
          <element name="status" type="typens:TransactionStatusCode"/>
          <element name="statusDetail" type="xsd:string"/>
        </sequence>
        <attribute name="objectId" type="xsd:ID" use="required" />
      </complexType>
```

This structure has the following elements:

- **messageCategory**: This is a notification type of either: Event, Document, or Transaction, as defined in the NotificationType enumeration.
- **messageName**: The name of the notification message.
- **status**: The current status of the object.
- **statusDetail**: This is a string that contain human readable description of the status.
- **objectId**: This is the unique ID associated with the notification object. It should be the TransactionId for transaction notification, DocumentId for document notification and Event Name for event notification. This attribute uses the XSD:ID type, which requires an underscore (_) as the first character of the string. Please refer to (http://www.w3.org/TR/xmlschema-2/#ID) for more information on the XSD:ID type definition.

3.8 Node Fault Detail Type

The version 2.1 WSDL specifies the structure of a specific fault message, NodeFaultDetailType, which must be sent whenever an error condition is raised by a Node. The SOAP 1.2 Specification requires that this fault message be the child of the

Detail element within the root Body element of the SOAP message (see example below).

The ErrorCode element is a list of error codes that are specific to the Exchange Network (see table 2 below), and the Description is a text string description of the error condition.

Although the **Detail** element is optional in the SOAP 1.2 specification, all Network Nodes should provide fault detail information whenever possible. The Exchange Network defines the **NodeFaultDetail** element, which is a child of the SOAP Detail, as follows:

```
      <element name="NodeFaultDetailType">
        <complexType>
          <sequence>
            <element name="errorCode" type="typens:ErrorCodeList"
minOccurs="1" maxOccurs="1" nillable="false"/>
            <element name="description" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="false"/>
          </sequence>
        </complexType>
      </element>
```

The Network Exchange Protocol version 2.1 list of predefined Exchange Network error codes is shown in Table 4.

Error Code	Description
E_UnknownUser	The user could not be found in the specified domain. The error occurs when the user is not registered in the domain or the user ID is incorrect.
E_InvalidCredential	The user credential is invalid. The error occurs when the security system could not verify user supplied password or digital certificate.
E_TransactionId	The supplied transaction ID could not be found. In methods such as GetStatus or Download, a transaction ID might be required and it must match a previous transaction.
E_UnknownMethod	The requested method is not supported. This indicates that the name of the web method is not defined in the Node Functional Specification.
E_ServiceUnavailable	The requested data service or web service is undefined or not supported The service provider returns this error when the request element in Query or Solicit, or the webMethod element in the Execute call is not recognized.
E_AccessDenied	The operation could not be performed due to insufficient privileges. The user must be authorized by the node administrator or dataflow administrator in order to access the service.

Error Code	Description
E_InvalidToken	The security token is invalid or not issued by a trusted security provider
E_TokenExpired	The security token has expired. A security token has a lifespan, and it must be used within the time period.
E_FileNotFound	The requested file could not be located.
E_ValidationFailed	XML schema or schematron validation error. This could occur when validation of request message or payload failed.
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, it must be in the range between 0 and maxRows returned from the service provider
E_FeatureUnsupported	The requested feature is not supported.
E_VersionMismatch	The request is a different version of the technical specification. This occurs when the namespace of the request message does not match the service provider's.
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 recognized or supported It is usually an indication of incorrect dataflow name.
E_InvalidParameter	One of the input parameters is invalid. The service provider should indicate the offending parameter name in the fault details.
E_AuthMethod	The authentication method is not supported.
E_Unknown	An unknown or undefined error has occurred. The error code might be used to indicate an unexpected condition, however the fault detail should contain additional information or a description of nature of the error.
E_QueryReturnSetTooBig	The result set specified is too large to return asynchronously The caller should set the maxRows to a smaller number, or use the Solicit method instead.
E_DBMSError	An internal database error occurred which prevents processing the request. This is typically a server fault.

Error Code	Description
E_RecipientNotSupported	The recipient functionality is not supported. Although the error code is defined here, it is highly recommended a node support the recipient feature whenever possible. It should treat the recipient parameter as non-critical and proceed even it is not implemented.
E_NotificationURINotSupported	The NotificationURI functionality is not supported. Usage of this error code should be limited. A node should use 'the best effort' for transaction notification as needed.

Table 4: 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. The value of this element should be a detailed description of the specific nature or cause of the error.

An example SOAP message with NodeFaultDetail:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
        <Code>SOAP-ENV:Sender</Code>
        <Reason>Invalid User</Reason>
        <Detail>
          <NodeFaultDetail
xmlns="http://www.exchangenetwork.net/schema/node/2">
            <ErrorCode>
              E_UnknownUser</ ErrorCode >
              <Description>
                Authentication failed; please check your userId and password.
              </Description>
            </NodeFaultDetail></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 that the default namespace for fault detail is:

http://www.exchangenetwork.net/schema/node/2

representing the XML namespace of this specification.

3.9 NotificationURI Type

The version 2.1 WSDL specifies the structure for a **NotificationURI** type. This type is used to specify a NotificationURI for the Solicit and Submit. This type includes a

structure of specific events, the NotificationTypeCode, which can be used to control when the notification address should be messaged.

The **NotificationURIType** is defined as follows:

```
      <complexType name="NotificationURIType">
        <simpleContent>
          <extension base="xsd:string">
            <attribute name="notificationType"
type="typens:NotificationTypeCode" use ="optional"/>
          </extension>
        </simpleContent>
      </complexType>
```

NotificationURI should contain one, and only one URI.

4 Network Service Interfaces

Network services defined in this specification are classified into four (4) major abstract interfaces:

Interface	Methods	Description
Data Submission	Submit, Notify, Download	A group of methods for sending arbitrary documents to a service provider and retrieving the results.
Data Publishing	Query, Solicit, Download	A set of methods for information retrieval from a service provider. The interface provides a framework for performing database queries.
Service Invocation	Execute, Download	A set of methods for offering services other than data services. This is a generic extension framework for adding extra services without redesigning and implementing a web service layer.
Administration	NodePing, GetServices, GetStatus	Methods for network-wide coordination and management.

Table 5: Network Service Interface Models

Figure 1 shows a static Unified Modeling Language (UML) diagram of interfaces in a network node.

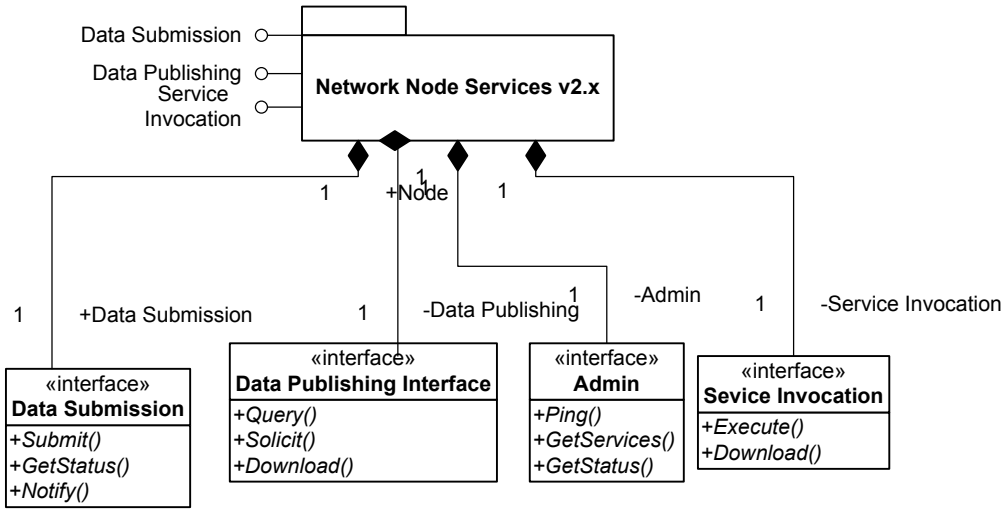

Figure 1: Static UML Diagram for Network Node Services

5 Message Encoding and MTOM Attachment

SOAP Message Transmission Optimization Mechanism (MTOM) is a W3C standard that defines a way of sending arbitrary binary data as an attachment without base64 encoding. Relying on XML Optimized Packaging (XOP), binary content is transmitted as it is on the wire using a very popular MIME multipart/related serialization of SOAP message parts.

In this encoding style, the SOAP message is transmitted as the first MIME part, followed by one or more attachments, referenced from the SOAP message body. Although MTOM supports attachments, the serialized message may contain only the SOAP message body, with no attachment at all.

In version 2.0 of the Node Functional Specification, MTOM encoding was only required for those web methods that need attachment support, while plain text encoding was used for all other methods. The mixed encodings of Text and MTOM have proven to be difficult in implementations and prone to interoperability issues among different toolkits.

This section discusses a set of requirements in message encoding that ease implementation and maximize interoperability between node clients and service providers.

5.1 Node Server MTOM Behaviors

In some very popular implementations of web service standards, MTOM encoding is either all or nothing by default. In other words, MTOM (in the same endpoint) is applied to all methods during the process of encoding response messages (if enabled) or some custom encoder has to be implemented to deal with mixed encodings.

To reduce implementation difficulties and align with industry support, all nodes MUST apply MTOM encoding to all response messages defined hereafter in this specification, regardless of the existence of attachments. The behavior applies to SOAP Fault messages as well.

A Node SHOULD accept and process messages that are not MTOM encoded in an effort to tolerate various implementations of node clients and maximize interoperability.

In many situations, XOP will not be applied during the MTOM encoding since it only applies to attachments of base64Binary type (only utilized in the NodeDocumentType in this document). Therefore, the SOAP message body is the only MIME part in such situations.

Note that even if there is an attachment, its contents may not be extracted by XOP as a separate MIME part, but rather sent as embedded base64Binary inside the SOAP message. This is due to the fact that XOP may only be triggered when the size of an attachment is greater than a threshold (e.g.2kb)

5.2 Node Client MTOM Behaviors

A node client is the consumer of node services, and the initiator of service requests. All node clients conforming to this specification MUST send MTOM encoded request messages, regardless of whether an attachment exists. This is, in fact, the default behavior of many toolkits.

A node client SHOULD handle both plain text or MTOM encoded response messages. The response tolerance allows a client to work with many v2.0 and v2.1 nodes. Our study indicates that this is also the default behavior in both .NET 3.0+ and Java implementations. The encoding style of a response message is largely transparent on the application level.

6 Interoperability Considerations

6.1 SOAPAction

SOAPAction is an HTTP header required by the SOAP 1.1 specification. It has been deprecated by the standard body in the SOAP 1.2 specification. However, there are still three variances of SOAPAction in some implementations that cause interoperability issues among Network Nodes:

1. SOAPAction in the HTTP Header
2. action in the Content-Type header as part of the media type.
3. wsa:action in the WS-Addressing header.

To simplify implementations and maximize interoperability, the following requirements are provided in this specification:

- Client applications MUST either include an empty value for SOAPAction in the HTTP header or not send SOAPAction at all.
- Node implementations MUST NOT require the presence of SOAPAction, action or wsa:Action nor use them as a dispatch mechanism. However, a node SHOULD tolerate the presence of SOAPAction and continue to process the request whenever possible.

6.2 Handling of WS-* and Other Web Service Standards

There are many web service standards and implementations that are defined and supported by various standards bodies. These include standards such as WS-Addressing and WS-Reliability and are often collectively referred to as WS-* standards, The WS-* standards can complement, overlap, and compete with each other. As such, the Exchange Network adopts the following principles with regard to the use of the WS-* standards:

- Minimal Dependency: The Exchange Network should be based on a minimal set of stable standard technologies that provide the most value. The minimal dependency strategy allows us to build fully functioning nodes without incurring the additional development/testing cost of supporting other specifications. It also reduces potential interoperability issues associated with various implementations of these standards across different platforms.
- Maximum Flexibility: Being neutral does not infer that these technologies are not to be used or supported. Exchange partners can make private arrangements for supporting these standards such as WS-Addressing, WS-Reliability or other WS-* standards as needed. The maximum flexibility strategy allows any node to electively support a WS-* standard within the EN framework in the future.

The following rules apply to the use of the WS-* standards on the Exchange Network:

- Clients SHOULD NOT send WS-* standards in a request, unless the receiving Node is known to support one or more of these optional standards.

- Nodes MAY support the WS-* standards, but their use MUST be marked as optional for requesters.

7 Node Web Methods

The following section describes the behavior and interfaces of the Exchange Network service provider. 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

Optional methods to implement include:

- Execute

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.

7.1 Authenticate

7.1.1 Description

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

A security token is an opaque string that is meaningful only to the issuer or trusted partners. 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.
- Other user properties such as user group or IP address.

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.

All messages, including Authenticate, must be sent using the Secure Socket Layer (SSL) transport mechanism. 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 has proven to be very expensive (public key infrastructure [PKI] required) and difficult to implement. The security token scheme presented here offers a simple yet effective way of identification and authentication.

This specification does not define the specific method for authenticating users or validating credentials. Each node implementer is free to choose any available authentication process in the underlying operating system, as outlined in the Exchange Network Protocol document. It is the responsibility of the node operator to choose a secure authentication process. Although password authentication is widely used in Network Node 1.1, it is strongly recommended that other, more secure authentication mechanisms such as key or certificate-based authentication, be supported in the version 2.1 nodes.

The Network Authentication and Authorization Service (NAAS) provides complete authentication and authorization services for the Exchange Network. All Exchange Network Nodes are encouraged to use the NAAS for authorization and authentication whenever possible.

7.1.2 Definition

Authenticate messages are defined below:

```
<element name="Authenticate">
  <complexType>
    <sequence>
      <element name="userId" type="xsd:string"/>
      <element name="credential" type="xsd:string"/>
      <element name="domain" type="xsd:string" nillable="true"/>
      <element name="authenticationMethod" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="AuthenticateResponse">
  <complexType>
    <sequence>
      <element name="securityToken" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
```

Where Authenticate is the request message; AuthenticateResponse is the response. The definition indicates that the Authenticate request message consists of four (4) arguments: **userId**, **credential**, **domain**, and **authenticationMethod,** all of type string. The response message contains a single string variable named **securityToken**, which contains the security token.

7.1.3 Arguments

- **userId:** the user ID of the person or system. It is recommended that an email address be used as the user ID in the Exchange Network.
- **credential:** the user's credential for accessing the network services. It could be a password, a password digest, or a secure authentication key. The **credential** could be an empty string in case of certificate authentication where the credential, the X.509 certificate, is provided inside the signature.
- **domain:** this parameter is optional. It is used for supporting multiple user identity stores or federated identity management systems. The default **domain** is for Exchange Network users is *default*. A node that supports multi-domain authentication should provide the name of the **domain** to Exchange Network Partners if appropriate.
- **authenticationMethod:** specifies which authentication methods are to be used. The **authenticationMethod** parameter could contain one of the following values:

Authentication Method	Description
Password	The credential parameter contains a password or a secure authentication key (SAK).
Digest	The credential parameter contains a digest (sha1) of the user's password.
Certificate	The credential parameter is empty, but the message is signed and a certificate is included in the signature
Token	The credential contains a security token issued by a trusted partner. This usually means that the user has already been authenticated by another security provider.

Table 6: Authentication Method Descriptions

A network node may elect to support WS-Security for authentication using certificates. The user must sign the authentication message and insert WS-Security headers in the SOAP request message. The Network Authentication and Authorization Services (NAAS) provide centralized authentication services, including WS-Security, available to all network nodes.

7.1.4 Return

Upon successful authentication, the service provider returns a SOAP message with a security token that is placed in **securityToken**. The security token 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 if the user record could not be found. The error code must be *E_InvalidCredential* when the system could not verify the supplied password or digital certificate.

7.1.5 Example

A typical request message is:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=" http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <SOAP-ENV:Body>
    <typens:Authenticate
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:userId>jsmith@example.com</typens:userId>
      <typens:credential>********</typens:credential>
      <typens:domain>galaxy</typens:domain>
      <typens:authenticationMethod>digest</typens:authenticationMethod>
    </typens:Authenticate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

and a positive response would be:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
…>
<SOAP-ENV:Body>
      <typens:AuthenticateResponse xmlns:typens="
      http://www.exchangenetwork.net/schema/node/2">
<typens:securityToken>34BjT34ngPRN2345INt</typens:securityToken>
      </typens:AuthenticateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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

7.2 Submit

7.2.1 Description

The Submit method provides a generic way of sending one or more payloads to a service provider. A payload is encoded as the document content (see definition of NodeDocumentType in section 3.3), and must be transmitted as an MTOM attachment.

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 indicating 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 should contain information for only one (1) dataflow at a time.

A new parameter, **recipient**, is introduced in this specification to support point-to-point secure data exchanges. A user can submit a document to another user by specifying the recipient's email address or node endpoint URI.

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.

7.2.2 Definition

```
      <element name="Submit">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="transactionId" type="xsd:string"/>
            <element name="dataflow" type="xsd:NCName"/>
            <element name="flowOperation" type="xsd:string" />
            <element name="recipient" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
            <element name="notificationURI" type="typens:NotificationURIType"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="documents" type="typens:NodeDocumentType"
minOccurs="1" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name="SubmitResponse" type="typens:StatusResponseType"/>
```

7.2.3 Arguments

The Submit method accepts six (6) 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 or a previous transaction (process report from a backend processor, for instance). All Exchange Network transaction IDs must begin with an underscore ('_') character.
- **dataflow**: The name of target dataflow. A dataflow identifier may contain sub-dataflow name if needed. The format of the **dataflow** with sub-dataflow should be:
  dataflow.sub-dataflow
  For example, a sub-dataflow Handler in the RCRA dataflow could be represented as:
  RCRA_v1_0.Handler
- **flowOperation**: A flow specific operation identifier. It indicates the specific processing for the document, as defined in the Data Exchange Flow Configuration Document (FCD).
- **recipient**[1]: An array of zero or more URIs representing the ultimate receivers of the submission. Each **recipient** item in the array should contain one node address URI or one email address URI. If **recipient** is a Node URI, the processed[2] submission package must be sent to the recipient Node using the Submit method. Specific details on how the Submit request to the recipient should be constructed is determined by dataflows and documented in the relevant Flow Configuration Document. If the recipient parameter contains a valid email URI, then the receiving Node must send an email containing at minimum: the transactionID of the submission, the receiving Node address, and the originating Node address. Additional details may be included, but are determined by dataflows and specified in the relevant Flow Configuration Document. This information must allow a properly authorized person to download the submission documents after receiving a valid transactionID via email.
- **notificationURI**: An array of zero or more URIs to which a status notification containing the processing status of a submission can be sent when the status of the transaction changes. The **notificationURI** parameter must contain either a valid email URI or a valid node address. An optional **notificationType** attribute may be specified to indicate the situations in which a notification message must be sent to the specified URI. If no **notificationType** attribute is specified, all messages relating to the transaction must be sent.

  If the value of this parameter is an email URI, the processing Node must attempt to send an email message which contains the ID, **statusCode,** and **statusDetail** of the transaction. If the value of this parameter is a Node URI, the Notify method will be

---

[1] More detail about specific use cases and functionality of the 'recipient' parameter can be found in the *Recipient Guidance and Best-practices* document.

[2] Node processing is determined by the dataflow specified in the 'dataflow' parameter and the 'flowOperation' parameter. Depending on how the service is defined in the Flow Configuration Document, processing may or may not alter the contents of the submission package.

called on the URI, including the transaction ID and the transaction status information.

- **documents**: One or more documents of type NodeDocumentType as described in section 3.3. Each document structure contains a single payload.

7.2.4 Return

The Submit method returns, when successful, a transaction ID, a transaction status code, and a description of the status. A major difference between the version 2.1 and the version 1.1 specification is the addition of transactionStatus to the return. This allows the option for synchronous transaction status messaging when the transaction is processed immediately. (e.g. the payload is relatively small and an informative transaction status can be returned immediately). However, nodes are required to return a transaction status regardless of whether the request is processed synchronously or asychronously. Nodes should simply return a status of *Received* if subsequent processing status information is not available immediately.

Whether or not a submission is processed immediately or at a later time, a transaction ID must be returned. It can be used to query status of the submission (see GetStatus method) later.

If the securityToken is invalid, insufficient, or expired, a node must return a SOAP fault message of type NodeFaultDetailType as described in section 3.8 with one of the following three error codes: *E_InvalidToken, E_AccessDenied,* or *E_TokenExpired*.

If the **recipient** parameter is unsupported, a SOAP fault message with an error code of *E_RecipientNotSupported* must be returned. Likewise, if the **NotificationURI** parameter is unsupported, an error of *E_NotificationURINotSupported* must be returned. If both parameters are unsupported, an error of *E_FeatureUnsupported* must be returned.

The service provider must return a SOAP fault message (Client Fault) if one of the payloads in the message could not be processed. The NodeFaultDetail element should contain additional error information.

7.2.5 Example

The following example shows a request message with one (1) referenced attachment. The payload is targeted to a dataflow called NEI.

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime">
  <SOAP-ENV:Body>
    <typens:Submit
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>csm:ZTyVjameqhpXZip0KFKwEfO57RKdQA,,.
</typens:securityToken>
```

```
      <typens:transactionId>uuid:_9w0BAQUFADBDMQswCQYDVQQGEw
</typens:transactionId>
      <typens:dataflow>NEI_v3_0.Point</typens:dataflow>
      <typens:documents DocumentId="f654c35c-f223-4787-a947-8787f532d3fe">
        <typens:DocumentName>NEI_DC_2005</typens:DocumentName>
        <typens:DocumentType>XML</typens:DocumentType>
        <typens:DocumentContent xmime:contentType='text/xml'>
MIICvTCCAiagAwIBAgIBZTANBgkqhkiG9w0BAQUFADBDMQswCQYDVQQGEwJJRTEP
MA0GA1UECBMGRHVibGluMSMwIQYDVQQDExpSU0EgVGVzdCBDQSAtIE5vIExpYWJp
bGl0eTAeFw0wNDAzMDIwMTIxNDZaFw0wNTAzMDIwMTIxN
        </typens:DocumentContent>
      </typens:documents>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Note that the document content is base64 encoded in the sample request message. All payloads, regardless of encoding type, must be sent using the MTOM attachment mechanism.

7.3 Download

7.3.1 Description

The Download method provides a means for retrieving documents associated with a transaction from a Node. In a typical asynchronous transaction, such as those associated with Solicit or Submit, the Download method is used to obtain results from a service request.

The ability to download documents makes mutual data exchanges possible. Any node in the Exchange Network can be a service provider and, at the same time, a service consumer. From the requestor'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 the requester access to documents. Generally, the Download method should only be available to users who have been given explicit permissions to invoke the service. Additionally, documents available through the Download method should have user and group level permissions that allow for granular security.

7.3.2 Definition

The request message:

```
      <element name="Download">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="dataflow" type="xsd:NCName"/>
            <element name="transactionId" type="xsd:string"/>
            <element name="documents" type="typens:NodeDocumentType"
minOccurs="0" maxOccurs="unbounded"/>
          </sequence>
```

```
        </complexType>
      </element>
      <element name="DownloadResponse">
        <complexType>
          <sequence>
            <element name="documents" type="typens:NodeDocumentType"
minOccurs="1" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
```

7.3.3 Arguments

The Download method takes the following parameters:

- **securityToken**: A security ticket issued by the service provider or a trusted security provider.
- **dataflow**: The dataflow identifier for the download operation.
- **transactionId**: A transaction ID for the submission. It must be the same transaction ID issued by the node (See the Notify method.)
- **documents**: An array of NodeDocumentType structures as described in section 3.3. This structure is used to specify what documents to download, using the DocumentName and documented attributes. The DocumentContents element must be empty in the request message, and should be ignored by service providers.

When the **documents** parameter is empty, the Node must return all documents associated with the supplied transactionId. When a document name is specified within the **documents** parameter, the node must return only the associated document.

For any valid transaction ID, the following predefined DocumentName attributes can be used to retrieve transaction reports, original documents, or some processed documents associated with the transaction. Nodes must have the capability to generate these four types of documents for all Data Exchanges. However if the specified DocumentName cannot be found, or does not exist for the transaction ID, the *E_FileNotFound* error must be returned. The specific content and formatting of each document is determined by each Data Exchange and should be documented in the relevant Flow Configuration Document.

DocumentName	Description
Node20.Report	Processing report if the transaction succeeded. These reports are flow specific and are defined in the relevant Flow Configuration Document.
Node20.Error	Error report if the transaction failed.
Node20.Orignal	The original document submitted by the user. The documentId attribute defines the

	specific document if there are multiple documents in the submission. All documents must be returned if no document Id is specified.
Node20.Processed	A processed document or a copy of record for the submitter.

Table 7: DocumentName Descriptions

7.3.4 Return

The response message contains an array of zero or more NodeDocumentType structures if successful. Documents composed of binary data must be transmitted as MTOM processed attachments with attachment type identifiers.

7.3.5 Examples

The sample message below shows a Download request for a report associated with the transaction 433612a7-83d3-4d2b-b3b4-05e3813d54bb:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime" >
  <SOAP-ENV:Body>
    <typens:Download
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>csm:3F4T322VxuPs23QrWspmR</typens:securityToken>
      <typens:transactionId>433612a7-83d3-4d2b-b3b4-
05e3813d54bb</typens:transactionId>
      <typens:documents>
        <typens:documentName>Node20.Report</typens:documentName>
        <typens:documentType>XML</typens:documentType>
        <typens:documentContent />
      </typens:documents>
    </typens:Download>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following request asks for all original documents associated with the transaction 433612a7-83d3-4d2b-b3b4-05e3813d54bb

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime" >
  <SOAP-ENV:Body>
    <typens:Download
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>csm:3F4T322VxuPs23QrWspmR</typens:securityToken>
      <typens:transactionId>433612a7-83d3-4d2b-b3b4-
05e3813d54bb</typens:transactionId>
```

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

7.4 Query

7.4.1 Description

The Query method is a function in the Data Publishing interface. The method is intended to run one of 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 may be unique to a particular node.

Another case where positioned-fetch must 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. Nodes are encouraged to support positioned-fetches (paged queries); however, due to varying system capabilities, some nodes may not be able to support this functionality. If a node cannot support positioned fetches, it must return a SOAP fault with an error code of *E_FeatureUnsupported* when rowId is greater than 0.

One of the new features introduced in the version 2.1 Query method is the dataflow parameter, which provides additional semantics to the data services requests and further limits possible data service name collisions.

Another change in the node 2.1 Query method is that query parameters are defined as elements with Name, Type, and Encoding attribute. This allows service providers to perform parameter binding without ambiguities. However, it should be noted that parameters with encoding types other than type string and XML may not be supported in all nodes, and thus should not be used for core business processes unless a Trading Partner Agreement (TPA) is established.

The successful response message contains a ResultSetType structure, which not only contains the actual result XML document, but also the description of the result set.

7.4.2 Definition

The Query messages are defined by the following WSDL segments:

```
      <element name="Query">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="dataflow" type="xsd:NCName" />
            <element name="request" type="xsd:string" />
            <element name="rowId" type="xsd:integer"/>
            <element name="maxRows" type="xsd:integer"/>
            <element name="parameters" type="typens:ParameterType"
minOccurs="0" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
```

```
</element>
<element name="QueryResponse" type="typens:ResultSetType"/>
```

7.4.3 Arguments

The Query method requires the following arguments:

- **securityToken**: A security ticket issued by the service provider or a trusted security provider.
- **dataflow**: The name of the dataflow.
- **request**: The database query to be processed. This should be defined for each data exchange and listed in the corresponding Flow Configuration Document (FCD).
- **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 paging is not requested.
- **maxRows**: The maximum number of rows to be returned. Valid values are any number greater than 0, and -1. If maxRows is -1, the service provider must return the entire result set (and indicates that paging is not requested). If the request maxRow parameter is too large, or the maxRows parameter is -1, the node does not support paging, and the result set is too large to process synchronously, a SOAP fault message with an error code of *E_QueryReturnSetTooBig* must be returned. A node may return fewer results if the complete result set is smaller than the value specified in maxRows. The rowCount element in the response represents the actual number of records returned under such situations.
- **parameters**: An array of zero or more ParameterType structures (see section 3.6) for the information request. Note that Nodes are only required to support parameters of encoding type String and XML. All other encoding types are optional, and it is the responsibility of the requestor to verify that other parameter encoding types are supported by the receiving Node.

7.4.4 Return

The Query method returns a result set as the ResultSetType (See section 3.4) if successful. It must return a SOAP fault message when it fails. The fault detail element may contain an error code and/or an error description from the native database system.

The service provider must return a SOAP fault message of type NodeFaultDetailType (see section 3.10) with an error code type of *E_RowIdOutofRange* if the rowId is out of range of the whole result set. The service provider must return a SOAP fault message with an error code type of *E_InvalidParameter* if one of the parameters has an invalid type or encoding, or if a specified encoding is unsupported. The service provider must return a SOAP fault message with an error code type of *E_QueryReturnSetTooBig* if the requested result set is too large to process or return. In this case, the service consumer should retry the query as a Solicit request.

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

7.4.5 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:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime">
  <SOAP-ENV:Body>
    <typens:Query
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">

<typens:securityToken>csm:PPFDNFDFDFDSAFA1fdsafnP</typens:securityToken>
      <typens:dataflow>FRS_v20</typens:dataflow>
      <typens:request>GetFacilityByZipcode</typens:request>
      <typens:rowId>0</typens:rowId>
      <typens:maxRows>100</typens:maxRows>
      <parameter name="StateZipCode">20001</parameter>
    </typens:Query>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

When there are multiple parameters it is the service provider's responsibility to bind them based on names, rather than position.

7.5 Solicit

7.5.1 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.

The Solicit method is most often used to request a Query operation that may be too large to process immediately. However, the service provider may begin the Query operation immediately when the request is received, thus avoiding management of a transaction queue, or wait until a later time. 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 processed.

Once the requested operation is processed successfully, the service provider should update the status of the transaction to **Completed**. If the operation failed, 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 recipient. The location must contain a network node address that has an implementation of the Submit method. It is the requester's responsibility to check the transaction status and download the result if the recipient parameter is empty.

7.5.2 Definition

The Solicit messages are defined by the following WSDL segments:

```
      <element name="Solicit">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="dataflow" type="xsd:NCName" />
            <element name="request" type="xsd:string" />
            <element name="recipient" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
            <element name="notificationURI" type="typens:NotificationURIType"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="parameters" type="typens:ParameterType"
minOccurs="0" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name="SolicitResponse" type="typens:StatusResponseType"/>
```

7.5.3 Arguments

The Solicit method requires the following arguments:

- **securityToken:** A security ticket issued by the service provider or a trusted security provider.
- **dataflow**: The name of the dataflow.
- **request**: The data service to be performed. It is usually the name of a predefined service request.
- **recipient**[3]: An array of zero or more URIs representing the ultimate receivers of the results. Each **recipient** item in the array should contain one node addressURI or one email address URI. If 'recipient' is a Node URI, the processed result set must be sent to the recipient Node using the Submit method. Specific details on how the Submit request to the recipient should be constructed is determined by dataflows and documented in the relevant Flow Configuration Document.

  If the **recipient** parameter contains a valid email URI, then the receiving Node must send an email that must contain: the transactionID of the process, the receiving Node address, and the originating Node address. Additional details may be included, but are determined by dataflows and specified in the relevant Flow

[3] More detail about specific use cases and functionality of the 'recipient' parameter can be found in the *Recipient Guidance and Best-practices* document.

Configuration Document. This information must allow a properly authorized person to download the generated result set after receiving a valid transactionID via email.

- **notificationURI**: An array of zero or more URIs to which a status notification containing the processing status of the transaction can be sent when the status of the transaction changes. The **notificationURI** parameter must contain either a valid email URI or a valid node address. An optional **notificationType** attribute may be specified to indicate the situations in which a notification message must be sent to the specified URI. If no **notificationType** attribute is specified, all messages relating to the transaction must be sent.

  If the value of this parameter is an email URI, the processing Node must attempt to send an email message which contains the ID, **statusCode** and **statusDetail** of the transaction. If the value of this parameter is a Node URI, the Notify method will be called on the URI, including the transaction ID and the transaction status information.
- **parameters**: An array of zero or more ParameterType structures (see section 3.6) for the information request. Note that Nodes are only required to support parameters of encoding type String and XML. All other encoding types are optional, and it is the responsibility of the requestor to verify that other parameter encoding types are supported by the receiving Node.

7.5.4 Return

The method returns a StatusResponseType (see section 3.5) structure which contains the transaction ID and the current status details.

If the **recipient** parameter is unsupported, a SOAP fault message with an error code of *E_RecipientNotSupported* must be returned. Likewise, if the **NotificationURI** parameter is unsupported, an error of *E_NotificationURINotSupported* must be returned. If both parameters are unsupported, an error of *E_FeatureUnsupported* must be returned.

7.5.5 Example

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

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:typens="http://www.exchangenetwork.net/schema/node/2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
      <typens:Solicit
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
        <typens:securityToken> csm:3F4T322VxuPs23QrWspmR
</typens:securityToken>
        <typens:dataflow>FRS</typens:dataflow>
        <typens:recipients>jsmith@example.com</typens:recipients>
```

```
<typens:notificationURI>https://cdxnode.epacdxnode.net/node/2</typens:notific
ationURI>
        <typens:request>GetFacilityByZipcode</typens:request>
        <typens:parameters Name="USPSZipcode">20001</typens:parameters>
      </typens:Solicit>
    </SOAP-ENV:Body>  </SOAP-ENV:Envelope>
```

Note that the requester asked the service provider to send a notification message to a node at `https://cdxnode.epacdxnode.net/node/2` at the time when the results are ready or when the request failed. It is the requestor's responsibility to download the results from the node.

7.6 Notify

7.6.1 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. 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.
- **Status notification:** A service provider sends a message to a requester to provide the current status of a submission or service request.

7.6.2 Definition

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

```
      <element name="Notify">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="nodeAddress" type="xsd:string"/>
            <element name="dataflow" type="xsd:NCName"/>
            <element name="messages" type="typens:NotificationMessageType"
minOccurs="1" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name="NotifyResponse" type="typens:StatusResponseType"/>
```

7.6.3 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 the notification messages.
- **messages:** An array of notification messages. All messages contained in a single Notify message must originate from the same dataflow. Please see the definition of NotificationMessageType in Section 3.7 for details.

7.6.4 Return

The returned XML structure, if processed successfully, is of the ResponseStatusType which contains a transaction ID, a status code, and a string description of the status for the notification.

The returned transactionId in the response should be the same transaction ID in the request message if supplied by the caller.

7.6.5 Examples

The example below shows a document notification. In this case, one (1) file is available for the service provider to retrieve later.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime" >
  <SOAP-ENV:Body>
    <typens:Notify
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>
            csm:3F4T322VxuPs23QrWspmR
      </typens:securityToken>
      <typens:nodeAddress>
            https://cdx.epacdxnode.net/services/v20
      </typens:nodeAddress>
      <typens:dataflow>FRS</typens:dataflow>
      <typens:messages ObjectId="_307c5169-80b1-4231-a3ae-9dc6ed70d4f1">
        <typens:messageCategory>Document</typens:messageCategory>
        <typens:messageName>RefreshData</typens:messageName>
        <typens:status>Completed</typens:status>
        <typens:statusDetail>A FRS document for loading is ready for
download.</typens:statusDetail>
      </typens:messages>
   </typens:Notify>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Note that the ObjectId in the notification message is the unique document ID for retrieving the documents with the Download method.

The following example shows an event message, perhaps to announce the unavailability of a new data set:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime" >
  <SOAP-ENV:Body>
    <typens:Notify
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>
            csm:3F4T322VxuPs23QrWspmR
      </typens:securityToken>
      <typens:nodeAddress>
            https://cdx.epacdxnode.net/services/v20
      </typens:nodeAddress>
      <typens:dataflow>FRS</typens:dataflow>
      <typens:messages ObjectId="_307c5169-80b1-4231-a3ae-9dc6ed70d4f1">
        <typens:messageCategory>Event</typens: messageCategory>
        <typens:messageName>FRS_Delta_Change</typens:messageName>
        <typens:status>Completed</typens:status>
        <typens:statusDetail>The FRS information on this node has been
changed.</typens:statusDetail>
      </typens:messages>
  </typens:Notify>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

This is very useful if the receiver does periodic data collections at a node. The provider could use this mechanism to send notification messages when the data has been changed.

The notify method could also be used for transaction status notification. A status notification message is similar to:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime" >
  <SOAP-ENV:Body>
    <typens:Notify
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>
            csm:3F4T322VxuPs23QrWspmR
      </typens:securityToken>
      <typens:nodeAddress>
            https://cdx.epacdxnode.net/services/v20
      </typens:nodeAddress>
      <typens:dataflow>AQS</typens:dataflow>
      <typens:messages ObjectId="_307c5169-80b1-4231-a3ae-9dc6ed70d4f1">
        <typens: messageCategory>Transaction</typens: messageCategory>
        <typens:messageName>AQS Transaction</typens:messageName>
        <typens:status>Completed</typens:status>
        <typens:statusDetail>An AQS transaction has been processed
successfully.</typens:statusDetail>
```

```
      </typens:messages>
    </typens:Notify>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

It indicates that a transaction with ID `_307c5169-80b1-4231-a3ae-9dc6ed70d4f1` has been finished successfully. In a multi-tier architecture where a node delegates to backend processor for document processing, this is an important mechanism for the backend system to notify the Node of status of the transaction. The DocumentContent element may contain an error report if the transaction failed.

7.7 Execute

7.7.1 Description

Node 2.1 has re-defined the Execute method. Execute method implementation is optional for Nodes. The Execute has been redesigned as a generic web service extension mechanism that allows a node to offer additional services. It can be used to extend a Network Node in the following ways:

- **Interface to new services:** Any new service that could not be offered through other node methods, such as Query or Submit, can be offered through the Execute method.
- **Gateway to legacy applications:** The Execute method can be used as the web service layer, or wrapper, for many internal applications, thus turning these legacy applications into web services. The Execute method essentially provides a web service wrapper to legacy applications in this scenario.
- **Proxy to other web services:** A node could use Execute as the proxy to external web services. It translates and forwards requests to other remote web services providers for processing. The key benefit of such services is that all node client applications can access a wide range of web services without incurring WS client development costs. The generic invocation capability is based on the facts that almost all programming interfaces can be mapped to function names and a list of parameters. In other words, they can all be modeled as standard procedure calls. The Execute method offers a mechanism to reach such programmable service components. Services offered through the Execute method must be broadly grouped into interfaces (for example, a set of outside web services described in a WSDL). An interface is composed of methods, which must be mapped to a single, component service. An appropriate analogy would be to the Dataflow/Operation paradigm used by Exchange Network Data Exchanges.

One of the main advantages of the generic web service invocation mechanism is that it encapsulates the complexity of the internal business logic and provides a consistent interface to consumers. When a new service is published through the Execute method, there should be no changes or programming requirements on the client side as long as the client “knows” how to call the Execute method. On the other hand, the generic

mechanism introduces new challenges to consumers. They will need to discover what web methods are available and how to bind the parameters at run-time. The GetService method (See section 7.9) provides this information dynamically.

For nodes using NAAS as the authorization service, access control policies can be applied to Execute similar to the Query method. The interfaceName should be treated as the dataflow name and the methodName mapped to the requestName in the policy settings.

7.7.2 Definition

The Execute messages are defined by the following XML Schema.

```
      <element name="Execute">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="interfaceName" type="xsd:string"/>
            <element name="methodName" type="xsd:string" />
            <element name="parameters" type="typens:ParameterType"
minOccurs="0" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name="ExecuteResponse">
        <complexType>
          <sequence>
            <element name="transactionId" type="xsd:string" />
            <element name="status" type="typens:TransactionStatusCode" />
            <element name="results" type="typens:GenericXmlType"/>
          </sequence>
        </complexType>
      </element>
```

7.7.3 Arguments

The Execute method accepts four (4) arguments:

- **securityToken**: An authentication ticket issued by the service provider or a trusted security provider.
- **interfaceName**: The name of the internal interface. This value is similar in concept to the dataflow parameter in Submit or Query, and should be used to indicate the desired functionality.
- **methodName**: The name of the method to be invoked within the interface.
- **parameters**: An array of zero or more ParameterType structures (see section 3.6) for the service call.

7.7.4 Return

The Execute method returns an XML structure that contains the following information:

- **transactionId**: An ID for the transaction, whether or not the transaction is synchronous or asynchronous. The transactionId will be used to query the status (GetStatus) or download the results for asynchronous transactions.
- **status**: The current status of the transaction.

results: The processing results for the request in the form of a GenericXMLType (see section 3.1). It should contain all the information from the ultimate service provider if the request is processed synchronously.

The node should forward SOAP Fault directly to the caller if the remote web service provider encountered error conditions.

7.7.5 Example

In the following example, we assume a node wishes to offer a web service that allows users to subscribe to a data change event, such as service description changes. When subscribed, the node sends an email message to all subscribers. The Subscribe service is defined as below:

InterfaceName	MethodName	Parameters	Results
DataChangeEvent	Subscribe	emailAddress: The subscriber's email address	A string description of subscription status.

In order to subscribe to the event, a requester may send a similar message to the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:Execute
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>csm:3F4T322VxuPs23QrWspmR</typens:securityToken>
      <typens:interfaceName>DataChangeEvent</typens:interfaceName>
      <typens:methodName>Subscribe</typens:methodName>
      <typens:parameter
name=”emailAddress”>jsmith@example.com</typens:parameter>
    </typens:Execute>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

A successful response message would be:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:Execute
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
```

```
      <typens:transactionId>433612a7-83d3-4d2b-b3b4-
05e3813d54bb</typens:transactionId>
      <typens:status>Completed</typens:status>
      <typens:results>
        <message xmlns="http://example.com/schema/subs/2">
          Thank you for subscribing the data change event. We will send a
message to you when our node configuration changes.
        </message>
      </typens:results>
    </typens:Execute>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7.8 GetStatus

7.8.1 Description

GetStatus is a method for transaction tracking. Once initiated, a transaction enters into different processing stages. The GetStatus method offers the client a way of querying the current state of the transaction. Note that GetStatus is used for querying the status of both asynchronous and synchronous transactions. A service provider must persistently store (log) transactional information for the following methods: Submit, Download, Query, Solicit, Notify and Execute.

7.8.2 Definition

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

```
      <element name="GetStatus">
        <complexType>
          <sequence>
            <element name="securityToken" type="xsd:string"/>
            <element name="transactionId" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="GetStatusResponse" type="typens:StatusResponseType"/>
```

7.8.3 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.

7.8.4 Return

The GetStatus method returns a StatusResponseType structure (see section 3.5) which contains status code and a string description of the current status if the operation is successful. A list of common status codes is defined below:

- Received: The transaction has been received by the Node but has not yet been processed or scheduled for processing.

- Processing: The transaction is currently being processed.
- Pending: Processing of the documents has not begun, and is either scheduled to be processed at a later time or is awaiting approval.
- Approved: The submission has been approved or certified if it needs approval. However, the documents have not been delivered to the receiver yet.
- Processed: The request/submission has been processed at the node. However, any payload associated with the transaction has yet to be delivered to the final recipient, usually a backend process.
- Completed: The transaction has completed, no further action will be taken on the request/submission.
- Failed: The transaction has failed, no further action will be taken on the request/submission. The requester should reinitiate the transaction after the problem is fixed.
- Cancelled: The transaction has been cancelled by the node administrator or an approver.
- Unknown: The status of the transaction cannot be determined at this time.

All Nodes must return one of the transactions statuses listed above. However, a Node should provide a mechanism to return additional, flow level, status information in the StatusDetail field of the StatusResponseType.

The method returns a SOAP Fault with an error code of *E_TransactionId* 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.

7.8.5 Example

A requester may send the following message:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:GetStatus
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">

<typens:securityToken>csm:4TmSris23MSQrRsT3492PPq</typens:securityToken>
      <typens:transactionId>8aa828c3-53a0-41ae-9760-
7d9f54158090</typens:transactionId>
    </typens:GetStatus>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A positive response could be:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:GetStatusResponse
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:transactionId>8aa828c3-53a0-41ae-9760-
7d9f54158090</typens:transactionId>
      <typens:status>Pending</typens:status>
      <typens:statusDetail>The submission is received and pending for
approval</typens:statusDetail>
    </typens:GetStatusResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7.9 GetServices

7.9.1 Description

The GetServices method allows requesters to find out the capability of a Node and discover how the services can be invoked. The type of services (or service categories) that can be queried includes:

- **AllServices**: A list of all service categories supported by the GetService method.
- **Query**: Predefined data services that can be used in the Query method.
- **Solicit**: Predefined data services that can be used in the Solicit method.
- **Execute**: Predefined other web services that can be used in the Execute method.

A node may choose to support additional service description categories (meta-data) when needed. To get a complete list of all service types, a requester can pass AllServices as the value of the serviceCategory element. The service provider must return a list of meta-description categories governed by the following schema definition:

Using GetServices, a requester can determine the capability of a node at runtime and proceed accordingly. The returned XML document contains detailed meta information that could be used for runtime binding of parameters. On the other hand, it allows the service provider to extend the services provided, (e.g., add a new data service), without changing the infrastructure. The smart invocation and easy extensibility can greatly enhance the overall usability, stability and capability of the Exchange Network.

7.9.2 Definition

```
<element name="GetServices">
  <complexType>
    <sequence>
      <element name="securityToken" type="xsd:string"/>
      <element name="serviceCategory" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="GetServicesResponse" type="typens:GenericXmlType"/>
```

7.9.3 Arguments

- **securityToken**: An authentication ticket issued by the service provider or a trusted security provider.
- **serviceCategory:** a string, which may be one of the following:
  - AllServices: A complete list of all service description categories that can be used as the value of the element. The possible values are:
  - Solicit: Predefined data request definitions supported by the Solicit method.
  - Query: Predefined data request definitions supported by the Query method.
  - Execute: A list of other web services and parameter definition supported by the Execute method.

  Since service providers may elect to provide additional categories of services, the list of service categories is not limited to the definition above. When automatically retrieving service definitions, the Exchange Network Discovery Services (ENDS) will get definitions in all categories using the AllService parameter value.

7.9.4 Return

The returned message contains an XML document of type GenericXMLType (see section 3.1) that contains the service and parameter definitions. The XML schema for the service and parameter description is defined at http://www.exchangenetwork.net/schema/ENDS/2/GetServices_v2.0.xsd.

If the serviceCategory is Query, Solicit, or Execute, the node must return a list of all predefined service requests as well as their parameters suitable for being used as the argument for runtime binding and execution of these methods.

7.9.5 Examples

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

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:GetServices
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken> csm:3F4T322VxuPs23QrWspmR</typens:securityToken>
      <typens:serviceCategory>ServiceTypes</typens:serviceCategory>
    </typens:GetServices>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The requester can call the method, using Query as the value of ServiceType, 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://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:GetServices
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:securityToken>csm:3F4T322VxuPs23QrWspmR</typens:securityToken>
      <typens:serviceCategory>Query</typens:serviceCategory>
    </typens:GetServices>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7.10 NodePing

7.10.1 Description

The NodePing method is a utility method for determining whether a node is accessible. A positive response from the node indicates that it is alive 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.

7.10.2 Definition

```
<element name="NodePing">
  <complexType>
    <sequence>
      <element name="hello" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="NodePingResponse">
  <complexType>
    <sequence>
      <element name="nodeStatus" type="typens:NodeStatusCode"/>
      <element name="statusDetail" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
```

Where the NodeStatusCode is an enumerate type defined as:

```
<simpleType name="NodeStatusCode">
  <xsd:restriction base="xsd:string">
    <enumeration value="Ready"/>
    <enumeration value="Offline"/>
    <enumeration value="Busy"/>
    <enumeration value="Unknown"/>
  </xsd:restriction>
```

```
      </simpleType>
```

7.10.3 Arguments

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

7.10.4 Return

The NodePing method returns a positive response in normal operational mode. It may return SOAP fault if the service is completely down. The service provider should return the service status codes in the positive response message.

Status	Meaning
Ready	The service is up and running.
Busy	The service is heavily loaded, it is preferable that the caller send request later.
Offline	Although the node service layer is up, but the backend is offline. All dataflow related requests will not be processed.
Unknown	The service status is unknown. Any other statuses that are not defined falls into this category.

Table 8: Service Status Codes

A node should also return the node product name and software version number (e.g. EN-Node v2.4146) in the statusDetail element.

7.10.5 Examples

A NodePing example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <typens:NodePing
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">
      <typens:Hello>there !</typens:Hello>
    </typens:NodePing>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A positive response from the node may be：

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <SOAP-ENV:Body>

    <typens:NodePingResponse
xmlns:typens="http://www.exchangenetwork.net/schema/node/2">

      <typens:nodeStatus>Offline</typens:nodeStatus>
```

```
    <typens:statusDetail>EN-Node v2.4146</typens:statusDetail>
  </typens:NodePingResponse>
</SOAP-ENV:Body>
```

8 Service Administration

8.1 Transaction Handling

The following table lists the information that should be tracked by Exchange Network nodes for each transaction in order to meet the minimum method requirements detailed in the previous section:

Name	Description
Transaction ID	A unique ID for a user request. It should be a UUID.
Method Name	The name of the web method that triggered the transaction.
Request Name	The name of data request for Solicit or Query, it should be the methodName for Execute.
Dataflow	The name of the dataflow.
Parameters	A list of parameters associated with the request.
User Name	The requester's NAAS ID.
ClientIp	Clients IP address.
Recipients	Ultimate receivers of the documents.
Timestamp	The time when the request is received.
Status	The transaction status.
ErrorMessage	Error message if the transaction failed.

Table 9: Transaction Tracking Minimum Elements

A node must also have the ability to track each document handled internally. The table below contains the following minimum information a Node should store persistently about each document.

Name	Description
Transaction ID	The transaction ID this document is associated with.

Document ID	A unique ID for the document. It should be a UUID.
Document Name	The name of the document as provided by the requester.
Document Type	The type of the document as provided by the requester.
Content Type	The content type in the MTOM attachment.
Status	The processing status of the document.
Timestamp	Time when the document is received.

Table 10: Document Tracking Minimum Elements

8.2 Logging

All network nodes must log transactions in a persistent storage area that retains the following information:

- Security Token of the document submitter
- time received
- transaction status

Exchange Network Nodes should provide the capability to track transactions by transaction ID or the NAAS ID of the document submitter. It is also recommended that a log that contains detailed processing steps be provided to assist debugging.

Appendix A - 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.
14. W3C Recommendation "SOAP Message Transmission Optimization Mechanism", Martin Gudgin, Noah Mendelsohn, Mark Nottingham and Herve Ruellan, 26 January 2005. (See http://www.w3.org/TR/soap12-mtom)