NATIONAL ENVIRONMENTAL INFORMATION EXCHANGE NETWORK

MISSISSIPPI DEMONSTRATED NODE CONFIGURATION

Version 1.1 Final

December 31, 2003

VERSION HISTORY

Version #	Implemented By	Revision Date	Reason
0.1	Tony Pruitt	06/16/2003	Draft submitted to ECOS.
0.2	Tony Pruitt	06/27/2003	QA Revisions
0.3	Tony Pruitt	06/30/2003	Additional QA Revisions
0.4	Melanie Morris	07/07/2003	Proof and Final Edits / Comments
0.5	Tony Pruitt	07/10/2003	Minor Revisions
1.0	Melanie Morris Tony Pruitt	07/15/2003	Minor updates to final document – This is the RELEASED version
1.1	Tony Pruitt	12/30/2003	Upgrade to CDX Version 1.1 Specifications This release includes a new Node Asynchronous Client application.

EXECUTIVE SUMMARY

The National Environmental Information Exchange Network (Exchange Network) is a new solution for sharing environmental data between states, tribes, the U.S. Environmental Protection Agency (EPA) and other partners. The Exchange Network is built upon the latest web services standards to allow the exchange of information between discreet and distinctive trading partners based on the standards defined in the Exchange Network protocol. Utilization of the Internet and standards-based exchange methodology allow partners to flow data from/to their own environmental information systems to/from other partners.

The Node Pilot Project, sponsored by the Environmental Council of States (ECOS) Information Management Workgroup (IMWG) proved the interoperable concept of building an Exchange Network using the emerging web services standards to exchange environmental data.

The Mississippi Department of Environmental Quality (MDEQ) implemented its integrated environmental information management system (enSite[1]) in October 2000.

Recognizing the shift to integrated environmental data, the Performance Partnership Agreement (PPA) between MDEQ and EPA includes an agreement to work toward using MDEQ's integrated system to provide data to EPA's national databases.

MDEQ's participation in the Node Pilot Project, its 2002 Network Readiness Grant award, and 2002 Challenge grant award position MDEQ for early participation in the Exchange Network for FRS, NEI and RCRAInfo data flows.

Additionally, MDEQ's integrated environmental data has reached a maturity level such that the database is ripe for sharing information via the Exchange Network.

MDEQ, the EPA and other environmental agencies will benefit by eliminating the time-consuming and labor intensive processes that must be used to exchange information, by improving the quality and concurrency of the information, and by utilizing the information to better perform their regulatory requirements and serve the citizens of the United States and the State of Mississippi.

Mississippi is well position to be on the forefront of modernizing the process of information exchange between all of its trading partners as the Exchange Network matures and additional environmental data flows are published.

[1] enSite utilizes TEMPO (Tools for Environmental Management Protection Organizations) software, a proprietary product developed and marketed by American Management Systems (AMS).

TABLE OF CONTENTS

1 INTRODUCTION

1.1 OBJECTIVES

The objective of the National Environmental Information Exchange Network (Exchange Network) Mississippi Demonstrated Node Configuration document is to provide a detailed description of the Mississippi Network Node (Network Node) software implemented by CIBER and the Mississippi Department of Environmental Quality (MDEQ) as part of the Exchange Network 1.0 Pilot Project Team. This document focuses on the Mississippi implementation of the Network Node, the Microsoft .NET application framework upon which the Network Node was built, the CIBER application and data architecture utilized to create the Node components, the Node Object Model, operational and developmental system requirements, installation procedures, and testing components. The intended audience is any state, tribe, territory, or other trading partner that is considering creating a Network Node using Microsoft .NET.

1.2 SCOPE

The Mississippi Demonstrated Node Configuration document describes the implementation specifications of the Network Node and the deployment requirements and directions.

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

See Section 1.2 of the Network Exchange Protocol [1] and the Network Node Functional Specification [2] for the terminology used in this document and other project artifacts.

1.4 REFERENCES

[1] National Environmental Information Exchange Network Network Exchange Protocol, Version 1.1, September 17, 2003.

[2] National Environmental Information Exchange Network Network Node Functional Specification, Version 1.1, September 17, 2003.

[3] Blueprint for a National Environmental Information Network Exchange, Network Blueprint Team, October 30, 2000, Amended June 2001.

1.5 OVERVIEW

This document provides an overview of the Exchange Network describing the problems that it addresses, the networks purpose, and the scope of this implementation of the network. Section 2 provides a high-level overview of the Exchange Network protocol, the Network Node functional specifications, and Mississippi's implementation of a Network Node. Section 3 describes the architecture framework (system, application, database) upon which the Network Node was built along with the .NET Node Server and .NET Developer Workstation system requirements. The Network Node Object Model and its associated .NET classes and .NET web methods are described in Section 4. The new Node Asynchronous Client application is described in Section 5. In Section 6, the Network Node installation procedures and configuration options are explained. Finally, Section 7 describes the testing tools that can be used to validate the operations of the deployed Network Node. The previous version of the document included an appendix describing the

Microsoft .NET environment that is used to build the Network Node. This section has been removed. Please refer to www.microsoft.com for information on the Microsoft .NET 2003.

2 EXCHANGE NETWORK

2.1 NATIONAL ENVIRONMENTAL INFORMATION EXCHANGE NETWORK

The National Environmental Exchange Network is a joint State and Environmental Protection Agency (EPA) project for sharing environmental data between States, Tribes, EPA and other parties with an interest in environmental data.

The Blueprint for a National Environmental Information Exchange Network [3] describes a practical vision for exchanging environmental data by applying the technologies and approaches that transformed the Internet into the standard point of access to information and public services. The key elements of this document are listed below.

- The core purpose of the Exchange Network is to utilize Internet technologies to change the way data is exchanged between the States and EPA.
- The Exchange Network design is based on four basic principles.
  - Stewardship of specific data will be established by mutual agreement between the trading partners.
  - Stewards are responsible for the quality and availability of their data.
  - Network Exchange members whose use of stewarded data necessitates the maintenance of local copies are responsible for the integrity and currency of the copies.
  - Network Exchange members agree to use the Network Exchange technology standards described in the Blueprint.
- The Exchange Network facilitates information exchange between "nodes" maintained by the participating partners using the Internet based on standardized data exchange templates and governed by trading partner agreements.
- Data exchange templates (DET) identify types of information required by a particular data set based on a predefined standard and expressed in XML Schema format.
- Trading partner agreements (TPA) are formal documents adopted by two or more trading partners to define the responsibility of each partner in the electronic exchange of information.
- The initial scope of the Exchange Network will focus on limited information already being exchanged between State, EPA and tribal partners on a formal basis but will expand to include a broad set of environmental information and trading partners.
- The Node will provide environmental data in an extensible markup language (XML) document using a standard DET and governed by a TPA.
- States, EPA and tribes established the National Environmental Data Standards Council to define the data standards used by the Exchange Network. These standards are used to build the DET.
- The technical infrastructure of the Exchange Network defines the software, hardware and protocols required for the successful exchange of information between Nodes. Basic Internet TCP/IP and HTTP protocols will be used to exchange XML documents between nodes. Secure HTTP (HTTPS) and Secure Socket Layers (SSL) will support the secure exchange of information.

Figure 2.1 illustrates the Exchange Network with EPA's CDX and several state nodes.

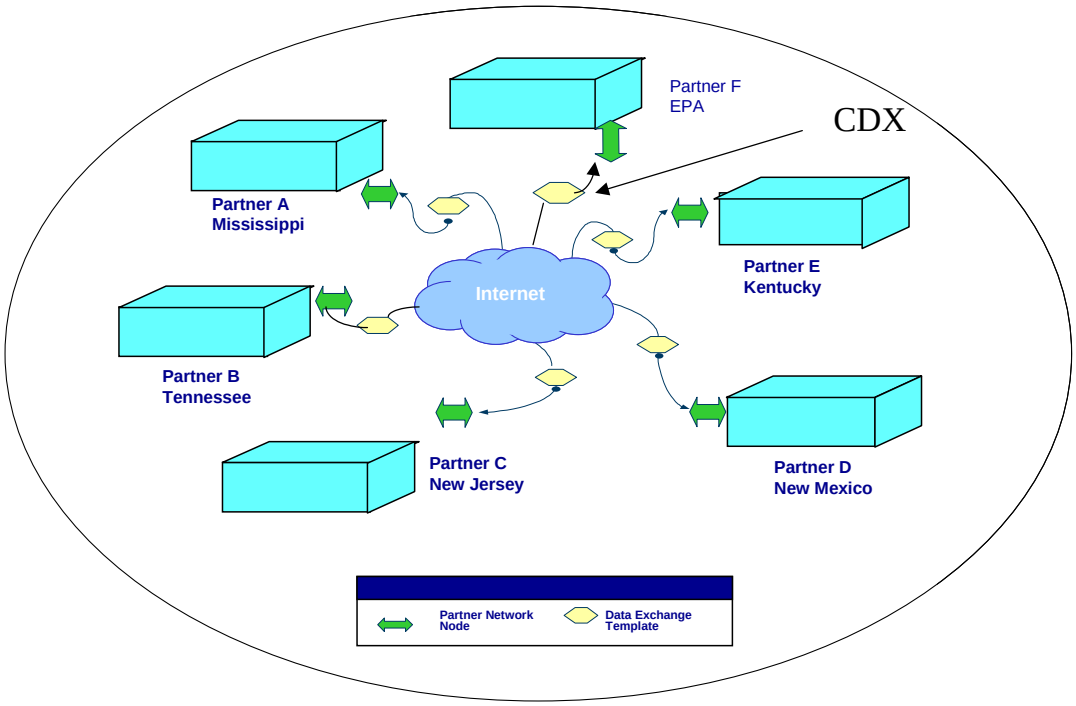

Figure 2-1: Environmental Network Exchange

2.2 EXCHANGE NETWORK PROTOCOL

The Exchange Network Protocol Version 1.1 [1] was released on September 17, 2003 to by the Exchange Network Node Project team. The protocol defines a set of rules that govern the generation and use of valid service requests and responses on the Exchange Network. The key elements of this document are listed below.

- The Exchange Network Protocol defines the components of the Web Services Architecture that forms a standard structure to the system. The architecture defines the basic web service components as a Service Provider, Service Requester or Service Registry. The architecture is extended to implement the Network Exchange by providing flow of operations services. These additional components are Data Exchange Template (DET) Registry, DET Repository, Flow Configuration Document (FCD) Registry, Service Description Repository, and DET Authority.
- The Exchange Protocol is based on a set of Web Services and Internet standards: Secure Socket Layer (SSL), Hypertext Transfer Protocol (HTTP), Simple Object Access Protocol (SOAP), Extensible Markup Language (XML), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). *Please refer to section 3.5 Web Services Standards in the Exchange Network Protocol document for more information.*
- Exchange Network Protocol network messages are encoded using the SOAP envelope/header/body structure within the basic HTTP request/response structure.

XML Payloads (Data) can be embedded within the body of a SOAP message or attached to the SOAP message.

- The Exchange Network Protocol defines the standards to utilize the Exchange Network UDDI and FCD Registry to publish and discover network services.
- The Exchange Network Protocol defines the network exchange services operations as follows:
  - **Authenticate** – Client Authentication Method
  - **Submit** – Client Send Document Method
  - **GetStatus** – Return Status of Last Transaction Method
  - **Query** – Query Database Method
  - **Solicit** – Asynchronous Query Method
  - **Execute** – Update Database Method
  - **Notify** – Document, Event, or Status Notification Method
  - **Download** – Retrieve Document Method
  - **Ping** – Check Network Availability Method
  - **GetServices** – Administrative Node Capability Method
- The Exchange Network Protocol defines the possible business process interaction scenarios between network partners and the procedures required to establish business process agreements between partners.
- The Exchange Network Protocol also defines the error handling approach and SOAP Fault Codes returned by the Nodes when an error occurs.
- XML Payloads, the data being transferred between partners, can be embedded with the SOAP message or attached to the message. Network Exchange nodes must support Direct Internet Message Encapsulation (DIME) in order to transfer attached payloads.
- The Network Authentication and Authorization Service (NAAS) provides centralized and federated user authentication to all the nodes on the Exchange Network. Each node will enlist the NAAS to authenticate users and validate tokens returned by the authentication process. The NAAS is essential to establishing a trusted relationship among the nodes.

2.3 NETWORK NODE

Also revised and released by participants in the Network Node Project on September 17, 2003 was the Network Node Functional Specification V1.1 [2] of the National Environmental Information Exchange Network. It is a detailed description of a Network Node's expected behavior and includes a description of the functions a Network Node will perform, how those functions are invoked, and specifies the expected output from the Network Node. The Network Node functional specification relies on the Exchange Network Protocol to define type of valid messages a Node expects to receive and process. The key elements of this document are listed below.

- The Network Node specification is consistent with the Exchange Network Protocol, Security Guidelines, and Network Registry Guidelines and operations.
- The Network Node architecture is consistent with the Exchange Network Protocol architecture.

- The Network Node logging requirements are defined.
- The Network Node logical object model is defined along with complete detailed description of each node web method.
- Rules for Node Validation to ensure full compliance with the specifications are defined.

2.4 MISSISSIPPI NODE IMPLEMENTATION

EPA has awarded Exchange Network Readiness grants to states, tribes and territories to establish Network Nodes and to develop environmental data flows for the Exchange Network. The Network Node project originally involved eight state environmental agencies, EPA, ECOS, and supporting contractors. The pilot node described in this document was implemented for the State of Mississippi Department of Environmental Quality (MDEQ) using the Microsoft .NET Platform, Microsoft Visual Studio .NET development environment, and Oracle 8i database. The Oracle database is part of MDEQ's centralized environmental management system, enSite.

This version of the Mississippi Network Node will exchange MDEQ Agency Interest (facility) data with EPA's Facility Registry System (FRS) and the FRS Version 2.2 Schemas. FRS has defined a set of XML schemas for the exchange of facility data on the Exchange Network. Please visit the FRS web site at http://www.epa.gov/enviro/frs for more information. Additionally, MDEQ has received grant funding to develop network flows to EPA's National Emissions Inventory (NEI) system, RCRAInfo, and Permit Compliance System (PCS) through IDEF. This version of the Mississippi Network Node has been designed to easily exchange any of the data flows listed above as well as new data flows in the future.

It has been CIBER's privilege as a member of the Exchange Network Pilot Project to participate in the development of the Exchange Network and the development of Mississippi's Network Node. As the author of this document, CIBER will demonstrate how Mississippi's Network Node was implemented using the Microsoft .NET environment and Oracle database. As the following sections will describe, CIBER follows proven object-oriented and architectural centric methodologies to develop quality application software and services utilizing highly experienced consultants. While this document demonstrates using Microsoft .NET to create a Network Node, CIBER has the experience and expertise to easily cast the node architecture into a Java environment hosted on IBM WebSphere, BEA WebLogic, or Oracle 9i AS application servers. In additional to Oracle 8i database, CIBER also has extensive expertise with Microsoft SQL Server and IBM DB2 databases. If you have any questions or would like to discuss services provide by CIBER, please contact Tony Pruitt at (770) 564-2099 Ext. 3 or by email at tpruitt@ciber.com.

3 NODE ARCHITECTURE

3.1 MICROSOFT .NET FRAMEWORK

The following comments are first hand observations about the Microsoft .NET environment from Tony Pruitt, the Mississippi Network Node architect. Mr. Pruitt's comments are intended to give his impressions of .NET without the typical marketing language and hype used to describe most commercial products.

"As a long time user of Microsoft Visual Studio, the new Visual Studio .NET (VS.NET) Integrated Development Environment (IDE) provides a robust and complete development environment for creating stand-alone applications, web applications using ASP.NET, web services applications, and .NET managed components that can be deployed to a separate application server and referenced by other applications."

"VS.NET now supports all the major object oriented behaviors such as inheritance. This is quite a change for developers with Visual Studio 6 background since prior Visual Basic development was a mix of traditional procedural coding and Object-Oriented Application Development (OOAD). If you have experience with Java or any other true OOAD environment, then VS.NET will be quite easy to adopt. I especially like the new C# language since it is syntactically very similar to Java. This is not surprising since both Java and C# are syntactically derived from the C language. The Mississippi Network Node was developed with the C# language. Unless you have existing VB applications, I highly encourage the use of C# since it was built as a true object-oriented language from the ground up."

"As a solution architect, I can create an application object model with a modeling tool, such as Rational Rose, and create web applications for either .NET or J2EE environments. Products like Rose make this even easier by creating actual .NET or Java code classes from the UML Class models."

"Since Microsoft along with IBM has been one of the major drivers of SOAP and XML standards, .NET offers support for all the published standards including DIME and UDDI. In order to get DIME support, you must download and install the Web Services Extension 1.0 Service Pack 1 .NET add-on package. Two of the features I found valuable was the ability to create abstract classes (for inheritance) from any WSDL file and to create data classes from XML Schema files, such as the FRS Version 2.2 schemas."

".NET also includes the ability to test web services from HTML Forms without any coding of these forms as long as the method does not include complex data types such as arrays. Using this capability, I created a web service to test the underlying components of the Node."

".NET also includes the Data Provider for Oracle that maps directly to the Oracle Net Client instead of using an ODBC layer between the application and Oracle Client. This offers much greater performance and allows .NET developers access to the Oracle database directly from the IDE. Given the appropriate permissions, the developer can create Oracle objects, browse and update tables, etc. Of course this capability is included out of the box for Microsoft SQL Server. You do need to download this add on from Microsoft."

"You can develop ASP.NET Web Forms just like you developed Windows Forms in prior versions of Visual Studio. While our team would rather code these forms to optimize the generated HTML instead of using the drag and drop techniques, I do believe many developers will find that Web Forms can be used to quickly develop dynamic HTML Forms."

"With most new products, I usually will wait until the second or third release before using it to develop customer applications but I found very few problems with the first release of .NET. A .NET solution hosted on a similar platform will generally outperform the same solution hosted on any of the J2EE implementations at this time. There are many other .NET features that could be discussed, such as ADO.NET and its XML underpinnings, but I recommend you visit MSDN (http://www.msdn.microsoft.com/). This site is more technical oriented (with less marketing and every article is rated by the MSDN community)."

3.2 SYSTEM ARCHITECTURE

The system architecture represents the physical network infrastructure that currently hosts Mississippi's Network Node. As illustrated in ***Figure 3.1***:

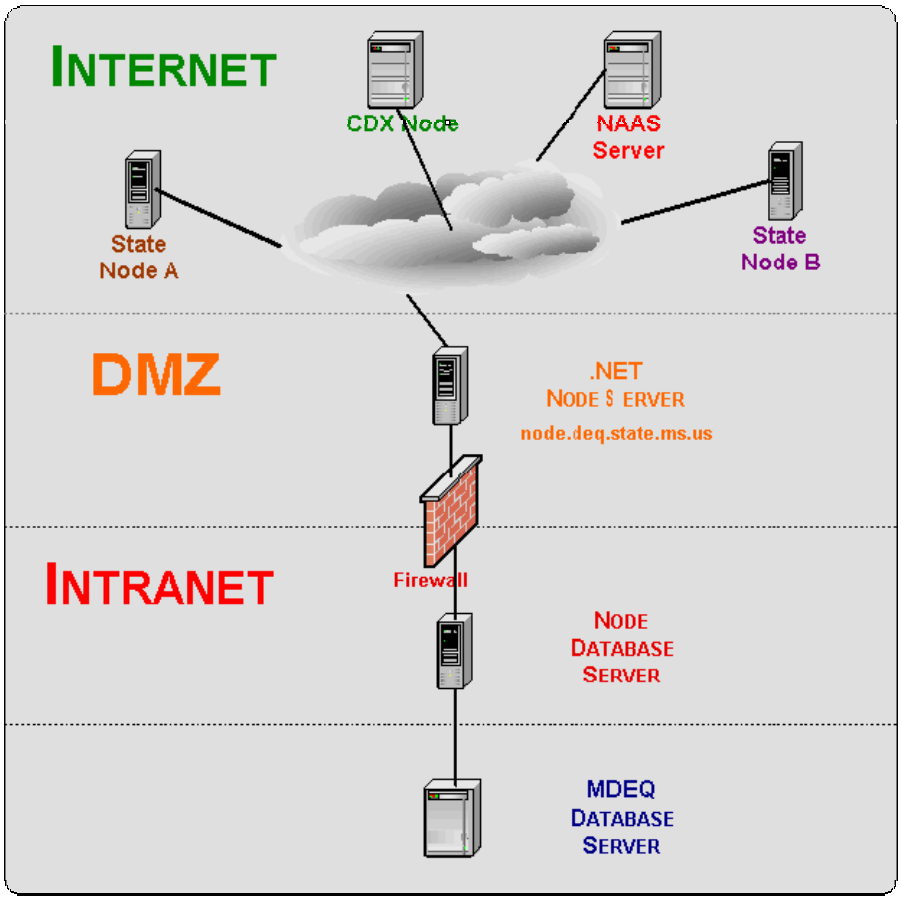

Figure 3-1: System Architecture

- Node Web Service Consumers (CDX and State Nodes) access the .NET Node Server via the Internet,

- The .NET Node Server resides in the MDEQ network DMZ and accesses the Node Database Server database as its primary database server,
- The DMZ is separated from the Intranet by a network firewall to protect the database from intruders and hackers,
- The Node Database Server contains a read-only copy of the MDEQ Database that is replicated from the production enSite database every night,
- The MDEQ Database Server can only be accessed from identified servers in the Intranet,
- While not illustrated, the MDEQ Database server may be segmented from the .Net Database Server and separated by another firewall, and
- The .NET Node Server accesses the NAAS Server to authenticate users and validate security tokens.

3.3 SYSTEM REQUIREMENTS

The following sections define the requirements for the .NET Node Server and the Visual Studio .NET Development Workstation. Please refer the to appropriate Oracle documentation for the Oracle Database system requirements.

.NET Node Server Minimum Requirements

Processor	Server (a computer working in a server capacity): 133-MHz Intel Pentium-class processor; 1.0-GHz or faster is recommend.
Operating System	The .NET Framework 1.1 Redistributable is supported on the following platforms: • Microsoft Windows® Server 2003* (.NET Framework 1.1 is installed as part of the operating system) • Windows XP Professional* • Windows 2000* **Notes:** ASP.NET Web applications and XML Web services can only be hosted on Windows XP Professional, Windows 2000, and Windows Server 2003 The .NET Framework Redistributable cannot be installed on 64-bit computers; Windows NT 4.0 Terminal Server is not supported
Memory	Server: 128 MB of RAM, 512 MB+ recommended
Hard Disk	110 MB of hard disk space required, 40 MB additional hard disk space required for installation (150 MB total), Large GB hard drives are highly recommend.
Display	800 x 600 or higher-resolution display with 256 colors
Input Device	Microsoft mouse or compatible pointing device
Other	• Internet Information Services (IIS) with the latest security updates must be installed prior to installing the .NET Framework. • Microsoft Internet Explorer 5.01 or later is required (click to get Internet Explorer 6.0). • Install the latest Windows service packs and critical updates from the Windows Update site. • Installation of the .NET Framework 1.1 Redistribution is required with the exception of Windows Server 2003. For more information, see the .NET Framework Downloads page. • Oracle 8i or greater client software must be installed and configured. The Oracle software is not provided with the DNC. Contact your local Oracle support staff to obtain a copy of the Oracle Client software. The Microsoft .NET Data Provider for Oracle is included with the NET Framework Version 1.1.

	• Microsoft Web Services Extension (WSE) Version 1 SP1 .NET Add-on is required to support DIME attachments. For more information, see the .NET Framework Downloads page.

Visual Studio .NET Developer Workstation Minimum Requirements

Processor	450-megahertz (MHz) Pentium II-class processor, 600-MHz Pentium III-class processor recommended
Operating System	Visual Studio .NET 2003 can be installed onto any of the following systems: • Microsoft Windows® Server 2003 • Windows XP Professional • Windows 2000 Professional (SP3 or later **required** for installation) • Windows 2000 Server (SP3 or later **required** for installation) Applications can be deployed onto the following systems: • Windows Server 2003[1] • Windows XP Professional[1] • Windows XP Home Edition • Windows 2000 (Service Pack 3 recommended)[1] • Windows Millennium Edition (Windows Me) • Windows 98 • Microsoft Windows NT® 4.0 (Service Pack 6a required) • Windows 95 (using Microsoft Visual C++® .NET) [1] *ASP.NET Web applications and XML Web services can only be hosted on Windows XP Professional, Windows 2000, and Windows Server 2003.*
Memory	• Windows Server 2003: 160 megabytes (MB) of RAM • Windows XP Professional: 160 MB of RAM • Windows 2000 Professional: 96 MB of RAM • Windows 2000 Server: 192 MB of RAM • 512 MB recommended
Hard Disk	• 900 MB of available space required on system drive, 3.3 gigabytes (GB) of available space required on installation drive • Additional 1.9 GB of available space required for optional MSDN Library documentation
Drive	CD-ROM or DVD-ROM drive
Display	Super VGA (1024 x 768) or higher-resolution display with 256 colors
Input Device	Microsoft mouse or compatible pointing device
Other	• Internet Information Services (IIS) with the latest security updates must be installed to test web applications and web services locally. • Microsoft Internet Explorer 5.01 or later is required (click to get Internet Explorer 6.0). • Install the latest Windows service packs and critical updates from the Windows Update site. • Microsoft Web Services Extension (WSE) Version 1 SP1 .NET Add-on is required to support DIME attachments. For more information, see the .NET Framework Downloads page. • Oracle 8i or greater client software must be installed and configured. The Oracle software is not provided with the DNC. Contact your local Oracle support staff to obtain a copy of the Oracle Client software.

3.4 APPLICATION ARCHITECTURE

The Mississippi Network Node architecture is designed as both a Web Service provider and a Web Service consumer of the state's environmental information. As a provider, Mississippi environmental data will be available to participating Network Exchange Nodes such as the EPA's CDX. As a consumer, MDEQ can request environmental data from other states and the EPA. While the initial pilot project will implement a provider of facility data, ECOS, the EPA and MDEQ envision a fully functional Network Node that will be a Provider and Consumer of all standardized environmental data available through the Exchange Network.

Figure 3.2 illustrates the Multi-Tiered Application Architecture model that the Mississippi Network Node was built upon followed by an explanation of each tiered element in the model.

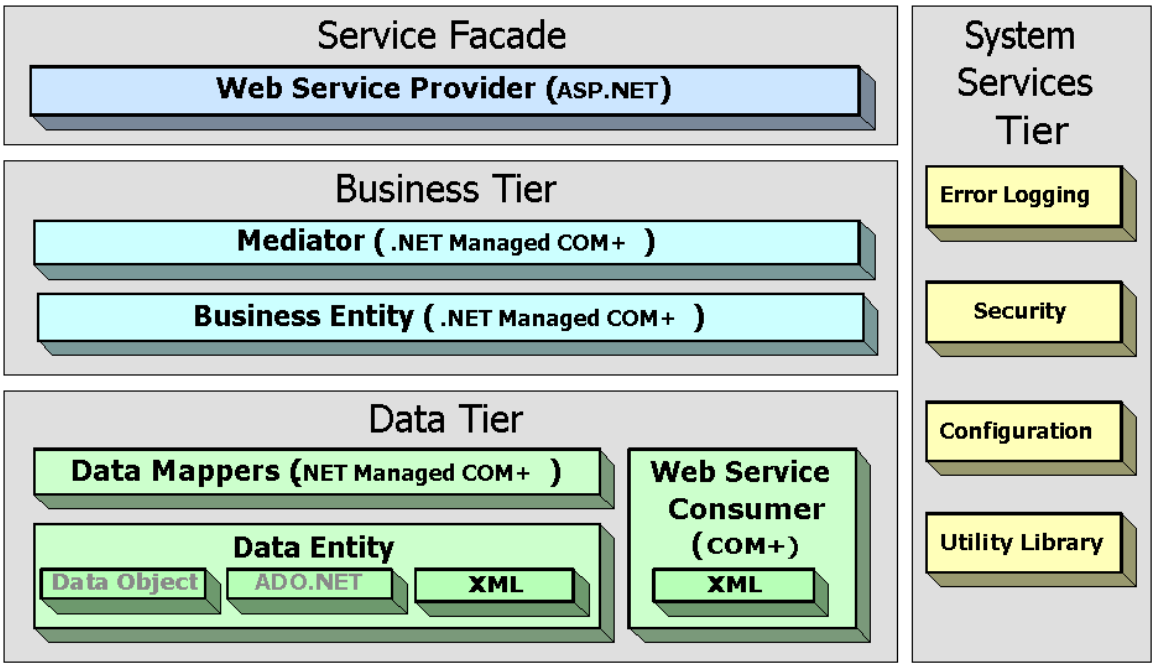

Figure 3-2: Application Architecture

- **Service Facade**

  The service façade exposes the functionality that should be visible to the outside world. The façade exposes the explicit interface definition and implements the service contract as published in the Web Services Description Language (WSDL).

- **Business Tier**

  The business tier components are .NET managed COM+ components (COM+) that perform business processes and logic. These components are independent of how the data they process is presented or stored. The business tier components are separated into the two layers in order to maintain better control over the business process, reduce the overhead of process changes, reduce the number of dependencies, simplify external communications, and simplify exception handling.

  - **Mediator** – Mediator business components are responsible for the business process. These components manage the steps necessary to carry out a business process. Mediator

components only manage the process and do not perform any business rules or data manipulation other than assembling the data for the business entity layer and presentation tier. Mediator components reference the data tier components to read or update the business data and send the results of the business process to the presentation tier components.

- **Business Entity** – Business Entity components are responsible for executing each of the individual steps of the business process. These components implement the business rules such as calculations, data conversions and validations. They do not store the business data; instead they reference data entities that store the data.

- **Data Tier**

  The data tier components are also COM+ components that communicate with the relational database or other data store. Requests are received from the business tier; the data tier interacts with the database, and returns the requested data in a form of data entities that can be easily processed by the business tier. Listed below are the two types of data tier components.

  - **Data Mapper** – These COM+ components are responsible for connecting to the database and mapping the data between the data entities and ADO.NET objects. Data Mapper components use ADO.NET to read and update the database and other data stores.
  - **Data Entity** – Data Entity components hold the logical data in a form that can be easily used by the business tier and services façade components. Data Entity components can come in three forms; (1) Serialized Data Object (COM+), (2) ADO.NET Dataset, or (3) XML Document. The Mississippi Network Node Data Mapper objects returns data in the form of ADO Datasets and the Mediator objects returns data as Serialized Data Objects that adhere to the Exchange Network standard XML Schemas utilized by the Web Services architecture.

- **System Service Tier**

  System Tier Components can be either ASP.NET or COM+ components. The system components provide common functionality that can be used by all the other tiers of the application architecture. Listed below are some of the common types of system tier components.

  - **Logging** – These components log all Network Node request activity to an application log on the middleware server.
  - **Security** – These components manages access to components and data based on a users role and profile.
  - **Configuration** – These components access system level configuration data that is stored in an XML configuration document.
  - **Utility** – These components provide functions to perform data conversions, data type validation, and other functions that are useful to the Network Node services.

3.5 DATA ARCHITECTURE

The data architecture is composed of the various objects that run within the context or directly access the relational database server, Oracle 8i. ***Figure 3.3*** represents the Mississippi Node Data Architecture.

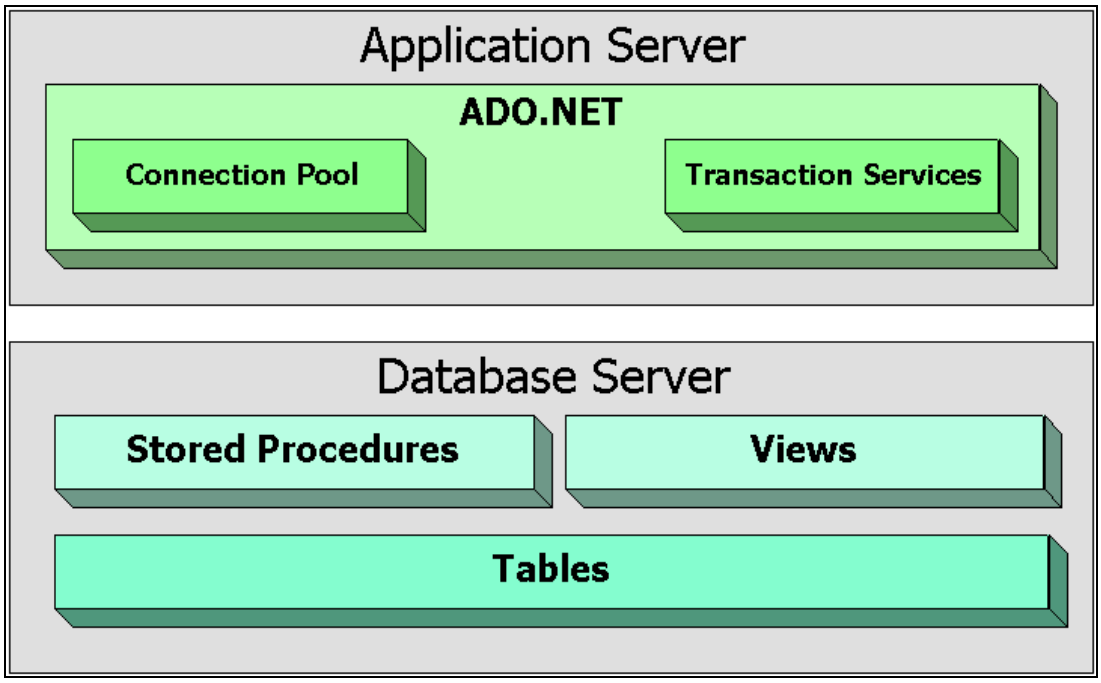

Figure 3-3: Mississippi Node Data Architecture

ADO.NET

ADO.NET is the set of interoperability and scalable data access services for the .NET Framework. ADO.NET is an evolutionary improvement to Microsoft ActiveX Data Objects (ADO) and ODBC built around n-tier development and XML. With Visual Studio .NET, developers program against objects, not tables and columns. ADO.NET features strongly typed programming, enabling developers to quickly write reliable data access code.

The centerpiece of ADO.NET is the data set. A data set is an in-memory copy of database data. A data set contains any number of data tables, each of which typically corresponds to a database table or view. A data set constitutes a "disconnected" view of the database data. That is, it exists in memory without an active connection to a database containing the corresponding tables or views. This disconnected architecture enables greater scalability by only using database server resources when reading or writing from the database. To accommodate the exchange of data between the multiple application tiers, ADO.NET uses an XML-based persistence and transmission format.

Connection Pool

One of the most expensive processes for any application is the time and resources it takes to connect to the database. In order to overcome this performance and resource bottleneck, a pool of connections is created for use by the Data Mapper components in the application architecture. ADO.NET provides integrated connection pooling for Oracle 8i and 9i database by using the Microsoft .NET Data Provider for Oracle, a separate component library that must be installed on the development and run-time environments. Each time a Data Mapper component attempts to connect to the database, ADO.NET will check its connection pool and use an available connection instead of creating a new one. The

connection is returned to the pool when the component closes the connection. All this is done without any special coding by the component developer.

Transaction Services

ADO.NET Transaction Services allows a series of insert, update, and delete procedures to be applied to the database server as a single transaction. If any of the SQL procedures fail, the database will rollback to its original state and will raise an error to the Mapper object that controls the transaction. ADO.NET transactional services also allow distributed transactions to multiple heterogeneous databases, such as Microsoft SQL Server and Oracle, to also be applied as a single transaction or rollback if any of the transactions fail.

Stored Procedures

The Data Mapper components rarely access the database tables directly for performance, security and changeability reasons. The most common approach is for these components to call SQL Stored Procedures to read or update the database tables. Stored Procedures are pre-compiled blocks of SQL code that are passed parameters and return row sets, XML document or singleton values to the calling components. SQL Stored Procedures offer the following advantages to other methods of accessing the database:

- The database table security access is limited to the Stored Procedures. The tables cannot be manipulated directly and are not visible to anyone but DBA and Network Administrators. While rules and constraints can be applied at the table level, Stored Procedures offer a greater level of control to prevent operations that may be legal from a SQL perspective but illegal from a business perspective.
- Stored Procedures are pre-compiled and pre-optimized to offer far greater performance than passing SQL statements to the database engine.
- Database developers can design, develop, and optimize stored procedures independent of the application development team. The component developers will only need to know how to use ADO.NET to call the Stored Procedure and access the returned dataset.
- Changes to the underlying database model can be made without impacting Data Mapper components unrelated to the change. The Stored Procedures can be changed to provide the same results to the components regardless of how the tables were modified.

Views

SQL Views are similar to stored procedures except that the component developer treats a view the same as a SQL table. Views are very useful for reporting and Ad Hoc queries in that they isolate the developer or tool from the data model and the proper use of relational Join statements. Views can also be used to partition data. By partitioning, the view will limit the data values its user sees based on a set of criteria coded into the view.

Tables

The application and portal physical database models are represented as SQL Tables in a relationship database. The application data elements, or columns, are organized into tables to store the actual raw data. In most cases, each table will have a primary index that uniquely identifies each row in the table. Foreign Keys may exist on tables that are related to a parent table. Column constraints can exist to enforce the values that are stored in a particular column.

Cached Data

One of the greatest performance improvements you can make to a website and its supporting database is to cache data frequently used but rarely changed in-memory of the web servers. This data is commonly configuration and drop-down list data used in web forms. The data is cached within ASP.NET Application or Cache context at web server startup time and is available to all users of the web server.

Microsoft .NET Data Provider for Oracle

The .NET Data Provider for Oracle is now included with the Microsoft .NET Framework version 1.1 to provide access to Oracle databases using the Oracle Call Interface (OCI) as provided by Oracle Client software. You must also have Oracle 8i Release 3 (8.1.7) Client or later installed and configured in both the development and run-time environments.

4 NODE OBJECT MODEL

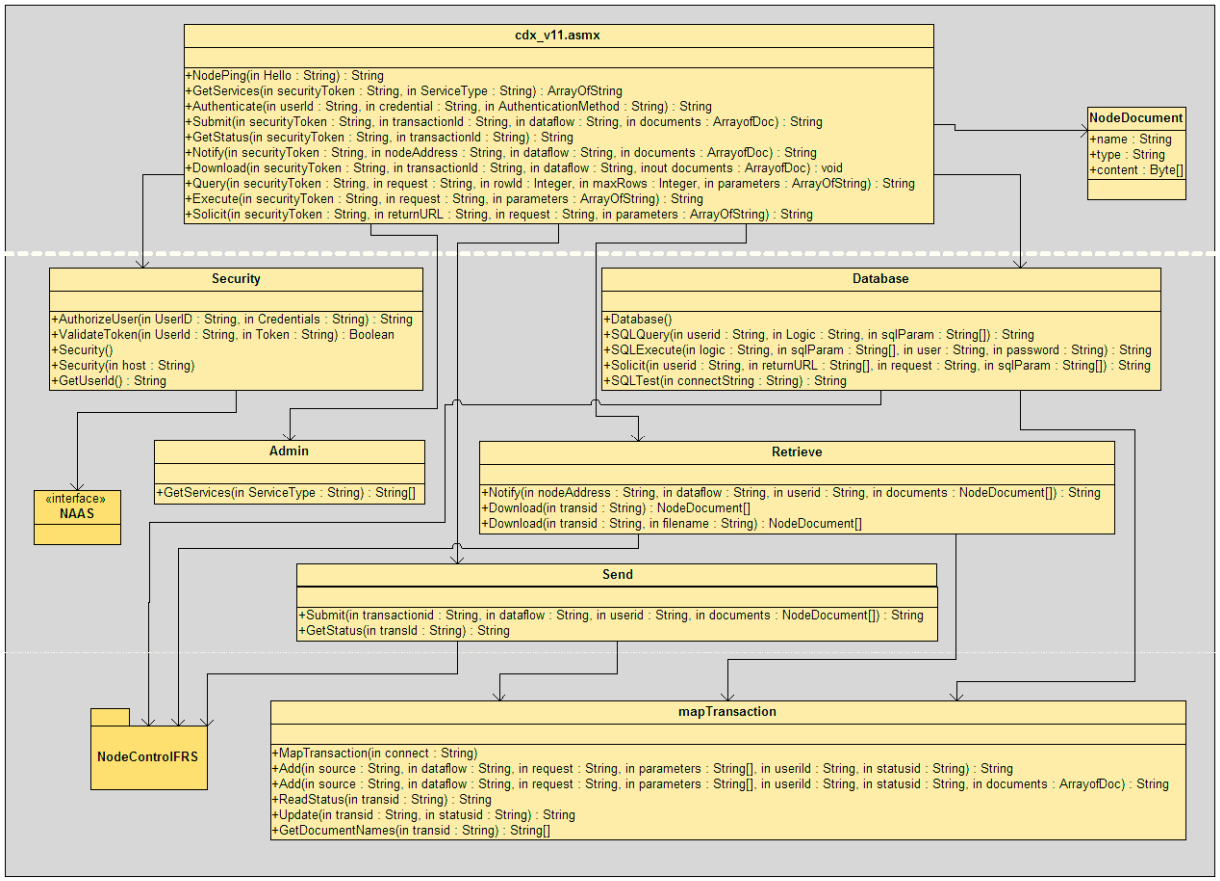

Figure 4-1: Mississippi Node Server Object Model

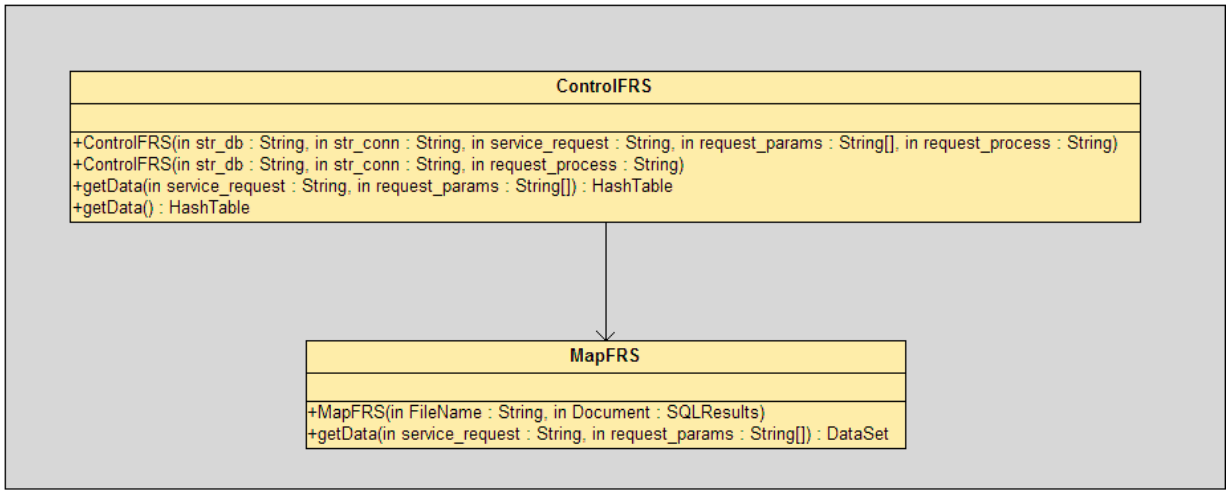

Figure 4-2: NodeControlFRS Object Model

4.1 NODE CLASSES

Figure 4.1 illustrates that the node object model is made of separate objects, or classes, that adhere to the application architecture discussed in *Section 3.4*. Each class is presented from the top of the object model starting with the service façade tier and followed by the business tier, data tier and finally the system tier. For any class method in the Mediator classes that references a Web Method, refer to *Section 4.2 Web Methods* for a detailed description. The service façade class methods utilize the associated mediator method to perform the logical process.

4.1.1 CDX_V11 Class

Class Type	**Service Façade**	
Description	CDX_V11 is a web service class that adheres to the Exchange Network Protocol Version 1.1 and Network Node Functional Specification Version 1.1.	
Methods	**Authenticate**	*Please refer to Section 4.2 Node Web Methods for a complete description of each web method. These methods conform to the Node Web Service Description (WSDL)*
	Submit	
	GetStatus	
	Notify	
	Download	
	Query	
	Solicit	
	Execute	
	NodePing	
	GetServices	

4.1.2 NodeDocument Class

Class Type	**Data Entity**	
Description	NodeDocument class is defined at the web service level as part of the CDX_V10 class. This class will contain document messages and reference information to attached DIME documents.	
Properties	**Name**	String
	Type	String
	Content	Byte Array (Base64Binary)

4.1.3 Security Class

Class Type	**System**

Description	Security class is responsible for authenticating Node users and validating security tokens. The Security class utilizes the Network Authentication and Authorization Service (NAAS) to perform it's authentication and security token validation task. The NAAS is hosted by EPA and is the recommended method for the Exchange Network security. All access to the NAAS is encrypted.	
Methods	**Security**	Class constructor method allows setting the NAAS Web Service URL.
	AuthorizeUser	Authenticates user access and returns a unique security token.
	ValidateToken	Validates security token passed to each of the Node web method.
	GetUserId	Returns the user id associated with the last ValidateToken method call.

4.1.4 Admin Class

Class Type	**Mediator Class**	
Description	The Admin class is responsible for network coordination and management.	
Methods	**GetServices**	This method will return the Service Types, Interfaces, Query Requests, and Execute Requests supported by the Node. *More…*

4.1.5 Database Class

Class Type	**Mediator Class**	
Description	The Database class is responsible for all database operations.	
Methods	**SQLQuery**	This method will query the database per a service request and return a XML document conforming to the FRS Schemas. *More…*
	SQLExecute	This method will execute a SQL Insert or Update operation that is applied to the Node database. *More…*
	Solicit	This method post a Solicited request to the Node database to be filled later by the Node Asynchronous Client. *More…*
	SQLTest	Performs a test connection to the Node database.

4.1.6 Retrieve Class

Class Type	**Mediator Class**

Description	The Retrieve class is responsible for event notification and document download.	
Methods	**Notify**	Post event notification messages to the Node database. *More…*
	Download	Downloads XML documents created asynchronously from the Node. *More…*

4.1.7 Send Class

Class Type	**Mediator Class**	
Description	The Send class is responsible for the submission of documents and for checking the status of service requests.	
Methods	**Submit**	This method will save (upload) attached documents to the Node. *More…*
	GetStatus	This method will return the status of a service request. *More…*

4.1.8 MapTransaction Class

Class Type	**Data Mapper**	
Description	MapTransaction class is responsible updating and querying the Node Transaction Log on the MDEQ Oracle database.	
Methods	**Add**	Inserts a transaction into the node transaction table and returns a transaction id.
	ReadStatus	Queries the status of a transaction based on a transaction id.
	Update	Update the status of a transaction based on a transaction id.
	GetDocumentNames	Returns a list of document name (XML files) based on a transaction id.

4.1.9 ControlFRS Class

Class Type	**Class Mapper**	
Description	ControlFRS class is responsible for converting the Datasets returned by the MapFRS class into FRS Schema Data Classes.	
Methods	**ControlFRS**	Class Constructor method sets database connection attributes.
	GetData	Retrieves FRS Data from MapFRS and converts into XML Schema Data Classes for use by the business tier.

4.1.10 MapFRS Class

Class Type	**Data Mapper**	
Description	MapFRS class is responsible for querying the MDEQ Oracle database for the Facility data into an ADO.Net dataset and returning the dataset to the ControlFRS class.	
Methods	**MapFRS**	Class Constructor method sets database connection attributes.
	GetData	Queries the Oracle database utilizing view objects per the service request and request parameters.

4.2 NODE WEB METHODS

The node web methods represent the public interface of the Node web service. A web method is "called" by a web service consumer in the same way a C# or Java component would call methods of a class except that in this case, the call is being made using the HTTP and SOAP protocols. The .NET Framework can also be configured to allow the web service methods to be accessed by either HTTP POST or HTTP GET request messages (these are commonly used in web applications written in Microsoft's Active Server Pages (ASP) or Java Server Pages (JSP)). The .NET Framework can also be configured to provide online HTML documentation of the web service methods. The following list of web methods was generated from Mississippi's Node server. Each web method name is hyperlinked to SOAP request and response message schema documentation on the Mississippi Network Node.

- NodePing
  NodePing method verifies the Network Node is available.
- Execute
  Execute method is intended for Non-Query service requests but is not implemented since all of these types of request must be processed by MDEQ's enSite system.
- GetStatus
  GetStatus returns the status of a specific transaction id.
- GetServices
  GetServices method returns list of Service Types, Interfaces, and Query Request supported by the Network Node.
- Submit
  Submit method allows an array of NodeDocument(s) and associated DIME attachments to be uploaded to the Network Node.
- Download
  Download method allows the downloading of documents from the Network Node.
- Notify
  Posts Event Notifications, Status Notifications, and Documents Notifications to the Network Node's transaction log.
- Authenticate
  Authenticates the web service consumer and returns a security token required for other operations.
- Solicit
  The Solicit method posts asynchronous service requests to the Network Nodes transaction log and returns a transaction id. An off-line asychronous process will fill the request.
- Query
  Query method extracts data from the database and returns an XML payload file.

The following is an example of the SOAP request and response message schema documentation for the Query web method.

```
POST /mscdx/cdx_v10.asmx HTTP/1.1
Host: node.deq.state.ms.us
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://node.deq.state.ms.us/mscdx/Query"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://node.deq.state.ms.us/mscdx/cdx_v10.wsdl"
xmlns:types="http://node.deq.state.ms.us/mscdx/cdx_v10.wsdl/encodedTypes"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <q1:Query xmlns:q1="http://www.ExchangeNetwork.net/schema/v1.0/cdx.xsd">
      <securityToken xsi:type="xsd:string">string</securityToken>
      <request xsi:type="xsd:string">string</request>
      <rowId xsi:type="xsd:integer">integer</rowId>
      <maxRows xsi:type="xsd:integer">integer</maxRows>
      <parameters href="#id1" />
    </q1:Query>
    <soapenc:Array id="id1" soapenc:arrayType="xsd:string[2]">
      <Item>string</Item>
      <Item>string</Item>
    </soapenc:Array>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://node.deq.state.ms.us/mscdx/cdx_v10.wsdl"
xmlns:types="http://node.deq.state.ms.us/mscdx/cdx_v10.wsdl/encodedTypes"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <q3:QueryResponse xmlns:q3="http://www.ExchangeNetwork.net/schema/v1.0/cdx.xsd">
      <return href="#id1" />
    </q3:QueryResponse>
    <q5:QueryResult id="id1" xsi:type="q5:QueryResult"
xmlns:q5="http://www.ExchangeNetwork.net/schema/v1.0/cdx.xsd">
      <results xsi:type="xsd:string">string</results>
    </q5:QueryResult>
  </soap:Body>
</soap:Envelope>
```

4.2.1 Authenticate Web Method

Description	The Authentication web method accepts user credentials and utilizes the NAAS to authenticate the user and return a security token. With the exception of the NodePing web method, all other web methods will require the web service consumer to include the security token in all other web method invocations.	
Parameters	**userId**	String
	credential	String
	authenticationMethod	String
Return	**securityToken**	String

4.2.2 Submit Web Method

Description	The Submit web method provides the means for another node to upload documents. Each NodeDocument in the documents array provides a reference to a DIME document attached to a SOAP request. The submission is logged to the Node database and the attached documents are saved to a configurable directory on the Node Server.	
Parameters	**securityToken**	String
	transactionId	String
	dataflow	String
	documents	ArrayofNodeDocument
Return	**transactionId**	String

4.2.3 GetStatus Web Method

Description	The GetStatus web method retrieves the status of a transaction. This is especially useful for tracking asynchronous service requests made with the Solicit web method. You must include the transactionId returned by the method you wish to track and the method will return a status of Received, Pending, Processed, Completed, or Failed.	
Parameters	**securityToken**	String
	transactionId	String
Return	**status**	String

4.2.4 Notify Web Method

Description	The Notify web method allows a web service consumer to send the node a document notification, event notification, or status notifications. All notifications are logged in the Node Transaction Log and will be used for reporting and asynchronous processes in the future.	
Parameters	**securityToken**	String
	nodeAddress	String
	dataflow	String
	documents	ArrayofNodeDocument
Return	**transactionId**	String

4.2.5 Download Web Method

Description	The Download web method provides the means for another node to download documents. This can be used to retrieve asynchronous request or documents that are published at regular intervals. Each NodeDocument in the documents array provides a reference to the DIME documents attached to the SOAP response. The method retrieves the documents created by the asynchronous process based on the transaction id returned by the Solicit method.	
Parameters	**securityToken**	String
	transactionId	String
	dataflow	String
	documents	ArrayofNodeDocument
Return		void

4.2.6 Query Web Method

Description	The Query method will query the Node database based on the Service Requests and parameters to return a set of FRS XML documents. The Solicit method should be used to exchange large FRS payloads. To get a list of service request supported by the Query method, execute the GetService web method with the "Query" parameter. Note that the RowId and MaxRows parameters are required but are not currently used. *Please refer to **Appendix C** for a list of Service Request currently supported and the parameters for each supported request.*	
Parameters	**securityToken**	String
	request	String
	rowId	Integer
	maxRows	Integer
	parameters	ArrayofString
Return	**return**	String

4.2.7 Solicit Web Method

Description	The Solicit method is very similar to the Query method except the request will be logged into the Node Transaction Log and the XML payload will be generated by an off-line asynchronous process. Using the returned Transaction ID, the status of the request can be checked with the GetStatus web method and the XML payload can be retrieved using the Download web method. The asynchronous process can also use the requester's Submit web method to send the payload to the requester based on the returnURL parameter. To get a list of service request supported by the Solicit method, execute the GetService web method with the "Solicit" parameter.	
Parameters	**securityToken**	Sting
	returnURL	String
	request	String
	parameters	ArrayofString
Return	**transactionId**	String

4.2.8 Execute Web Method

Description	The Execute web method is intended to allow for Insert and Update SQL statements or service requests and is optional. Since the node has Read-only access to the Node database, this method is not operational at this time.	
Parameters	**securityToken**	String
	request	String
	parameters	ArrayofString
Return	**rowAffected**	String

4.2.9 NodePing Web Method

Description	The NodePing web method functions much like a TCP/IP Ping method functions and simply test the availability of the Node. The node will return "Ready" response if it is operational. This is the only web method besides Authenticate that does not require a security token. While the node may be "Ready", the node database may not be available. To test the database availability, set the Hello parameter to "Connect".	
Parameters	**Hello**	String
Return	**state**	String

4.2.10 GetServices Web Method

Description	The GetServices web method is used to determine what services a Node offers. The ServiceType parameter will accept ServiceType, Interfaces, Query, Solicit, and Execute values. Of these, the Query and Solicit is the most useful by returning a list of service request the Query and Solicit web methods support. This allows additional service request to be added as they are developed.	
Parameters	**securityToken**	String
	ServiceType	String
Return	**Services**	ArrayofString

5 NODE ASYNCHRONOUS CLIENT

This section provides an overview of the Node Asynchronous Client (Node Client) application. The Node Client is a stand-alone command line executable that is intended to be executed on set intervals by the operating system timer. The requirements to install the Node Client are the same as the Network Node. The Node Client is responsible for the following operations:

- Fill solicit requests and submit the resulting XML documents to the requestor's Node,
- Fill unsolicited request and submit the resulting XML documents to the CDX Node.
- Submit XML documents previously created by the Node Client or XML Documents created by another system, and
- Notify requestors that a request has been filled and can be downloaded by the requestor.

Figure 5.1 illustrates the architecture of the Node Client and its reuse of the data mapping components used by Network Node server. The figure also shows the use of a network node proxy class to connect to any Exchange Network Node Version 1.1 implementation.

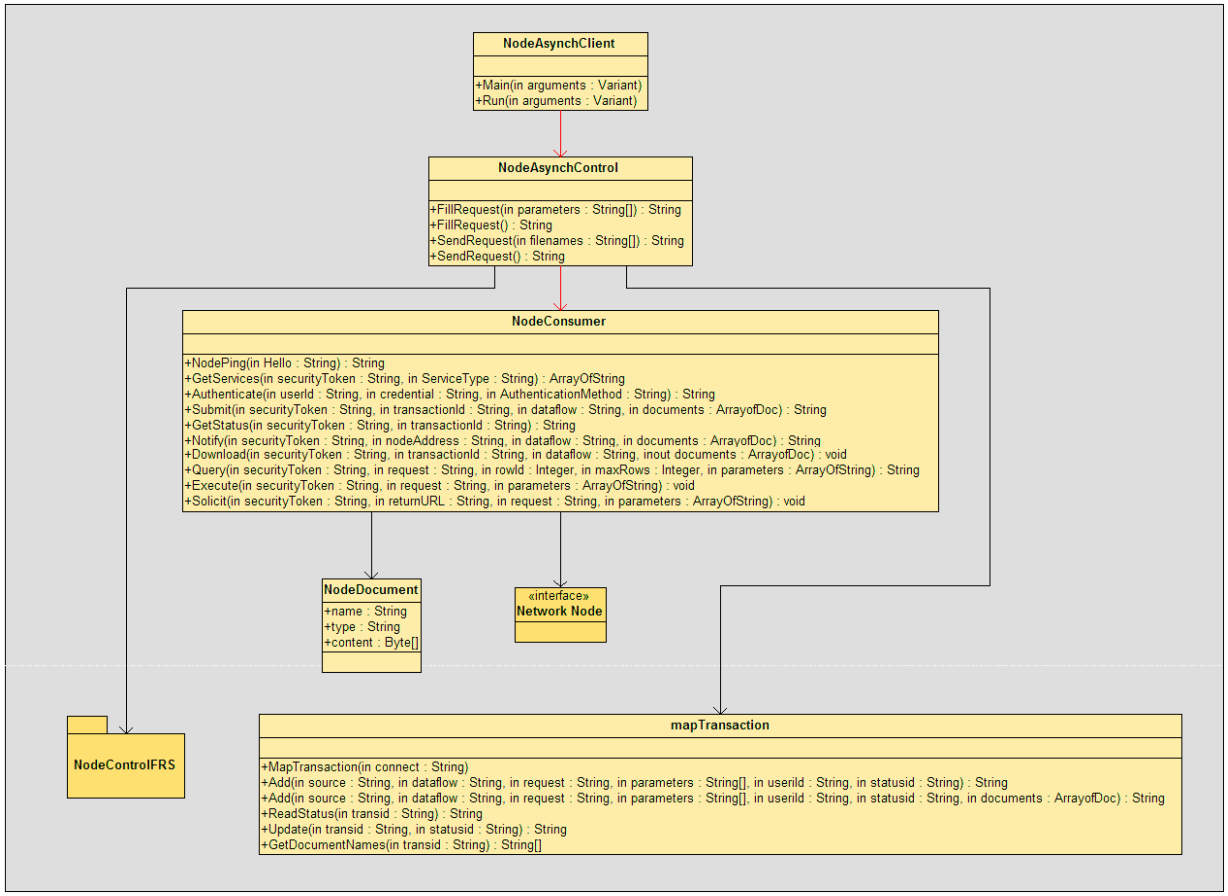

Figure 5-1: Mississippi Node Client Object Model

5.1 NODE CLIENT EXECUTION MODES

The Node Client is composed of two separate operations, or use cases, to meet the asynchronous needs of an automated Network Node. Command line syntax defines several modes of operation and associated attributes.

Syntax: <path>\NodeConsumerClient.Exe <mode> <mode attributes>

The following table list and describes each mode.

Solicit Mode	
Description	Fill Solicit requests and submit request payloads to the requesting Node.
Attributes	None
Syntax	*<path>\NodeConsumerClient.exe S* This example will return XML payloads for all outstanding solicit requests posted to the Node database.
Unsolicited Mode	
Description	Fill Unsolicited requests and submit request payload to the CDX Node. Use the Node's GetServices method to get a list of solicit request supported by the Node.
Attributes	• Service Request – The service request directs the node on what data to return (select a facility name, county, state id, city, etc.) • Requests Parameters – These are the parameters required by the service request to filter the data by.
Syntax	*<path>\NodeConsumerClient.exe U GetFacilityByName MS Acme* This example will return XML payloads for all facilities in Mississippi (*MS*) with a facility name equal *Acme*.
Fill Mode	
Description	Fill Solicit requests but do not submit payloads to the requesting Node. The requestor will either download (pull) the payload later or the Node Client will submit (push)
Attributes	None
Syntax	*<path>\NodeConsumerClient.exe F* This example will generate XML payloads for all outstanding solicit requests posted to the Node database and store them for later download or submission.
Transmit Mode	
Description	Transmit service request payloads previously filled to the requesting Node. These payloads could have been generated by another system and posted to the Node database.
Attributes	None
Syntax	*<path>\NodeConsumerClient.exe T* This example will generate XML payloads for all outstanding solicit requests posted to the Node database and store them for later download or submission.

Document Mode	
Description	Transmit payloads identified by the document names. This allows for transmission of a specific document (xml, zip, etc.) to the CDX Node.
Attributes	• Filename [1-n] – list each filename to submit. Please note that filenames cannot include spaces or directory pathname. The files must be stored in the Nodes Outbound Directory.
Syntax	*<path>\NodeConsumerClient.exe D FacilitySite.xml EnvironmentalInterest.xml* This example will submit the two files to the CDX Node.

6 NODE INSTALLATION

This section provides guidelines for building a server platform to host the Network Node, installing the .NET Framework and supporting packages. The following scenario steps through the process of setting up the node server at MDEQ's network operations center. The sections after the scenario will provide you with guidelines to deploy the .NET Network Node application in your environment. This document will not provide you with detail setup instructions to each of the installable components but will refer you to the setup instructions that accompany each component. The scenario may need to be modified based on the network standards of your environment. You will need the DNC CD loaded into the Node Server CD drive in order to install many of the following components.

Node Server Setup Scenario	
Prerequisites	An Oracle 8i or 9i database that will be accessible from the Node Server in the DMZ must already exist. *Please note that the Node application will install and offer minimal operations without a database.* A static IP address and optionally a DNS name must be defined for the Node Server.
Step 1	Install Windows 2000 Server along with the Internet Information Server (IIS 5.0) on the designated Node Server. Install Windows 2000 Service Pack 4, Internet Explorer 6 and the latest security patches and critical updates from Microsoft to complete the platform setup. If you are using Windows 2003 Server, install IIS 6.0 and any operating system or securities patches available from Microsoft.
Step 2	Install the .NET Framework 1.1 Redistribution package. If you are using Windows 2003 Server, the .NET Framework should already be installed.
Step 3	Install the Microsoft Web Extension Version 1 Service Pack 1 package.
Step 4	Install and configure the Oracle 8i Client software.
Step 5	This step would create the node database objects on the Oracle database. Because these objects would expose proprietary information, they are not included with the DNC. ***The Oracle database objects are available for TEMPO states upon request from MDEQ.***
Step 6	Configure IIS and create a virtual web directory for the Node application.
Step 7	Install the Network Node application
Step 8	Configure the Network Node application.

6.1 SETTING UP THE OPERATIONAL PLATFORM

The .NET framework can be deployed to several Microsoft operating systems as listed in the system requirements in Section 3.3. The MDEQ Network Node operates on a Microsoft 2000 Server running IIS 5.0 and is connected to the DMZ segment of the network. After installing the operating system and connecting to the network, start IIS and make sure you can access the Administration web site included with the installation package. The following table lists the system specification of the Mississippi Network Node Server.

Processor	1.2-GHz Pentium 4
Operating System	Windows 2000 Server with Service Pack 4
Machine Name	Node
URL	http://node.deq.state.us.ms
Memory	1 GB of RAM
Hard Disk	3 18GB hard disk drives in a Raid 5 configuration. The disk space is partitioned into 2 logical drives. The C: partition will contain the operating system and installed server applications while the D: partition will contain the web service application and associated content directories.
Other	• Internet Information Services (IIS 5.0) with the latest security updates • Microsoft Internet Explorer 6 • Norton Anti-Virus

6.2 MICROSOFT .NET FRAMEWORK 1.1 INSTALLATION

The Microsoft .NET Framework includes everything you need to run .NET Framework applications, including the Common Language Runtime, the .NET Framework class library, and ASP.NET including SOAP Web Services.

Skills Required: Installer must have administrative rights to install programs on the Node Server.

Installation File: Download the Microsoft .NET Framework Version 1.1 Redistribution installation package from the .NET Framework Downloads page.

Installation Instruction: Execute the installation file and follow the instructions of the installation program.

6.3 MICROSOFT WEB SERVICES ENHANCEMENTS 1.0 SP1 INSTALLATION

The Microsoft Web Services Enhancements 1.0 SP1 is required to support DIME attachment capabilities of the Node.

Skills Required: Installer must have administrative rights to install programs on the Node Server.

Installation File: Download the Microsoft .Web Services Enhancement 1.0 SP1 for .NET installation package from the .NET Framework Downloads page.

Installation Instruction: Execute the installation file and follow the instructions of the installation program.

6.4 ORACLE CLIENT SOFTWARE INSTALLATION

The Oracle 8i Client Software must be installed on any system that will access an Oracle database. Using the Net8 Assistant or Net8 Configuration Assistant, a net service name

entry must be created and configured properly to connect to the Oracle database. The following list is an example of the configuration information required:

- Net Service Name: NODEDB.NODEDOMAIN
- SID: NODEDB
- Protocol TCP/IP
- Host Name: nodedatabase.deq.state.ms.us
- Port: 1100

Skills Required: Installer must have administrative rights to install programs on the Node Server and be experience with configuring Oracle Clients. The installer will need guidance from the local Oracle DBA to determine the correct configuration values.

Installation File: Must be provided by the installing organization.

Installation Instruction: Refer to Oracle DBA.

Please note that the Oracle 8i Client may not install properly on Pentium 4 systems and installation of Oracle 8i and 9i Clients on the same system may cause a conflict. Your Oracle DBA should be able to resolve these problems if they arise.

6.5 ORACLE DATABASE CONFIGURATION

SQL objects must be added to the Node database in order to properly communicate with the Node application. The DDL scripts used by MDEQ to create the SQL objects and the FRS Data Mapping document that maps enSite/TEMPO data elements to the FRS schemas is available upon request to other TEMPO states.

The Node Transaction tables are defined in ***Appendix B***. These tables contain all the logged Network Node requests and status of the requests.

The DBA may wish to create a special Oracle user for the Node application to use or may choose to use an existing user. Whichever option is chosen, the user must be granted read rights to the view objects and reference tables and read-insert-update rights to the two Node Transaction tables.

Skills Required: Oracle DBA.

Installation Files: Provide Upon Request.

6.6 IIS CONFIGURATION

We recommend that you create an IIS virtual web directory to install the Node application into. The following installation instructions describe how this was done on the Mississippi Node Server.

Skills Required: Installer must have administrative rights to create Windows directories and IIS virtual directories.

Installation Instruction: Execute the following steps to configure the IIS server.

1. Create a windows directory for all .NET Web Application and Web Services: **D:\WebApps**
2. Create a windows directory for the NODE application: **D:\WebApps\Node**
3. Configure the Node application directory to a web virtual directory. This can be accomplished either by using the IIS Service Manager or the Windows File Manager. Since most users are familiar with File Manager, we will use this method.

    a. Navigate File Manager to the **D:\WebApps\Node** directory.
    b. Right click the **Node** folder and select the Properties option.

```
   c. Select the Web Sharing tab.
   d. Click the Share this folder radio button. The Edit Alias dialog
      will launch.
   e. Accept the Edit Alias default values and press OK.
   f. Press Ok and exit the properties dialog.
4. Create a directory to store all XML documents that is created from
   either an synchronous or asynchronous service request: D:\OutDoc
5. Create a directory to store all XML documents submitted to the Node
   via the Submit web method: D:\InDoc
```

6.7 NODE INSTALLATION

Once the Node virtual web directory has been created, you are now ready to install the Node application on the Node Server. Please refer to ***Appendix D*** for a list of files and directories installed into the virtual web directory.

Skills Required: Installer must have administrative rights to install programs on the Node Server.

Installation File: <*CD*>:\Install\ Application\Setup.exe.

Installation Instruction: Execute the following steps to install the NODE application

```
1. Run the Setup.Exe from the DNC CD.
2. The setup process prompts you for the Virtual Directory and Port.

   Virtual Directory: Node (This is the same directory created in Section
   5.5)
   Port: 80

3. Accept the default options in all other installation steps.
4. After installation, use a browser and navigate to http:/<your server
   name or IP address>/Node/cdx v10.asmx. This should display the web
   service documentation page. Select the NodePing Method and press
   Enter. The NodePing method should return "Ready" if the node was
   installed correctly.
```

6.8 NODE CONFIGURATION

The following items in the Web.config file located in the Node application web directory (D:\WebApps\Node\Web.config) will need to be configured for your environment.

- DSN – Data Source Name (Oracle Connection String)
- OUTDIR – Directory to store downloadable documents. This directory was created in Step 4 of Section 5.5.
- INDIR – Directory to store uploaded documents. This directory was created in Step 5 of Section 5.5.
- HOST – This key is used with the Network Authentication and Authorization Service (NAAS) to associate a user and security token with a specific node. Any unique secure identifier will work.

The following code example is the configuration entry that must be modified in the Web.config file. The **add** attributes of the <dsn> element for <appSettings> must be on one line, even though the following sample shows it split across multiple lines for readability.

```
<configuration>
  <appSettings>
```

```
    <add key="dsn" value="Data Source=NODEDB.DOMAIN;
      User ID=nodeuserid;Password=node password"/>
    <add key="outdir" value="d:\outdoc"/>
    <add key="indir" value=" d:\indoc"/>
    <add key="host" value="hostname2003"/>
  </appSettings>
</configuration>
```

6.9 NODE CLIENT INSTALLATION

The Node Client can be installed on either the Node Server or another computer. If the application is installed on another computer, this computer must meet all the requirements listed above with the exception of IIS and the Node application. Please note that sample batch execution files are included with the setup to execute the Node Client in different modes.

Skills Required: Installer must have administrative rights to install programs on the Node Server.

Installation File: <*CD*>:\Install\ Client\Setup.exe.

Installation Instruction: Follow the setup instructions to install the NODE CLIENT application

7 NODE TESTING

7.1 NODE TEST WEB SERVICE

The .NET Web Services includes the ability to self-test the Web Methods from the Node server if properly configured. As we mentioned earlier, the .NET web service can document the service in HTML format and if the Web.config is setup correctly, the less complex methods can be tested using HTML forms. Under normal circumstances, the Web.config file will be configured to not allow the use of HTML forms because they can present security problems for other Node implementations and slow down the discovery process since the WSDL is three times larger.

```
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <clear/>
        <add name="HttpSoap"/>
        <add name="Documentation"/>
      </protocols>
    </webServices>
  </system.web>
</configuration>
```

As you can see in this example, the <webServices> <protocols> section of the Web.config file is clearing all protocols and adding the HttpSoap and Documentation protocols. In order for the .NET web services HTML Form protocol to be activated for testing, the protocols section of the Web.config file must be deleted or commented out as the following example shows.

```
<configuration>
  <system.web>
    <webServices>
  <!—-PROTOCOL SECTION COMMENTED OUT FOR TESTING REMOVE BEFORE PRODUCTION
      <protocols>
        <clear/>
        <add name="HttpSoap"/>
        <add name="Documentation"/>
      </protocols>
    -->
    </webServices>
  </system.web>
</configuration>
```

This configuration will allow you to test the less complex methods (those that do not involve arrays) from an HTML form. Use your web browser and navigate to http://localhost/Node/cdx_v10.asmx and the documentation protocol will list the Node Web Methods. Click on NodePing method and you will be presented with an HTML form that will prompt you for a single parameter. The SOAP, HTTP GET and HTTP POST request and response message format are also displayed. Click the Invoke command button and the Web Method will be executed and the resulting response message displayed. For example, the NodePing method will display the following response.

```
<?xml version "1.0" encoding="utf-8" ?>configuration>
<string xmlns="http://node.deq.state.ms.us/mscdx/cdx v10.wsdl/
  literalTypes">Ready</string>
   <webServices>
```

The response to the NodePing web method is a simple **Ready** as shown above within the context of the response message.

To test the more complex web methods, a test web service is also included with the Node Installation. Use your web browser and navigate to http://localhost/Node/test_v10.asmx and the documentation protocol will list the Test Web Methods. The following section defines each test web method.

- Notify
  Test the Notify web method posting notification transactions to the Node Transaction Log.
- Query
  Test the Query method by prompting user for Service Request parameters.
- TransactionTest
  Test posting to Node Transaction Log.
- SQLTest
  Test Node database connection.
- Submit
  Test the Submit method to upload a document.
- TestToken
  Test the operation of the Network Authentication and Authorization Service (NAAS).

7.2 EPA NODE TEST TOOL

The EPA's Central Data Exchange (CDX) web site provides two node test tools for you to validate that the Node conforms to the Exchange Network Protocol [1]. The test tools can be accessed over the Internet by navigating to http://naas.epacdxnode.net from your browser. From this web site, follow the directions on the web site to utilize the test tools properly. The two test methods are described below. These methods will prompt you to enter a WSDL address file. The Mississippi Node WSDL address is http://node.deq.state.ms.us/mscdx/cdx_v10.asmx?wsdl.

- **Manual Test Tool**

  The manual tool allows you to test each of the Web Methods individually by providing your own parameters and return the result of the request in SOAP format. You must enter the WSDL Address File for your node on the main page to allow the test tool to invoke your web service.

- **Automated Test Tool**

  The automated test tool can be selected at the bottom of the test site page or by clicking here to link to the automated test tool (http://naas.epacdxnode.net/testcase.html). The

automated test executes a set of pre-defined test scenarios against each Web Method and returns the results as either “Passed” or “Failed”. You can also view the SOAP request and response message format from the results page.

APPENDIX A – TRANSACTION TABLE DEFINITION

NODE_TRANSACTION Table Definition					
PK	**Column Name**	**Type**	**Size**	**Null**	**Default**
*	NODE_TRANS_ID	VARCHAR2	30	N	
	NODE_TRANS_DT	DATE		Y	
	NODE_TRANS_SOURCE	VARCHAR2	100	Y	
	NODE_TRANS_DATAFLOW	VARCHAR2	10	Y	
	NODE_TRANS_REQUEST	VARCHAR2	50	N	
	NODE_TRANS_PARAM	VARCHAR2	50	Y	
	NODE_TRANS_UID	VARCHAR2	30	Y	
	NODE_TRANS_STATUS_ID[2]	CHARACTER	1	N	"0"
	NODE_TRANS_STATUS_NOTE	VARCHAR2	50	Y	
	NODE_TRANS_STATUS_DT	DATE		Y	Today

NODE_TRANS_DOC Table Definition					
PK	**Column Name**	**Type**		**Null**	**Default**
*	NODE_TRANS_ID	VARCHAR2	30	N	
*	NODE_TRANS_DOC_SEQ	NUMBER	3	N	
	NODE_TRANS_DOC_NAME	VARCHAR2	100	N	

[2] STATUS ID VALUES: 0 = OPEN, 1 = FILLED, 2 = SENT, 3 = RECEIVED, 4 = DOWNLOADED, 7 = REJECTED, 8 = FAILED, 9 = LOG ONLY

APPENDIX B – SUPPORTED SERVICE REQUESTS

Service Request	Parameters		Response Schema	
	1	2	Query	Solicit
`GetFacilityById`	State USPS	Facility Id	All*	All*
`GetFacilityByName`	State USPS	Facility Name%	Facility Site	All*
`GetFacilityByLocalityName`	State USPS	Locality Name	Facility Site	All*
`GetFacilityByCountyName`	State USPS	County Name	Facility Site	All*
`GetFacilityByStateId`	State USPS	<None>	Facility Site	All*
`GetFacilityByChangeDate`	State USPS	<None>	Facility Site	All*

**** = All FRS Schemas (****Facility Site, Environmental Interest, Alternative Name, Geographic Coordinates, Organization, Mailing Address, Individual, SIC, and NAICS Code****).***

APPENDIX C – NODE WEB DIRECTORY LISTING


bin

Microsoft.Web.Service.dll
Mscdx.dll
Mscdx.pdb
ControlFRS.dll
ControlFRS.pbd
OracleFRSData.dll
OracleFRSData.pbd
NodeOutputFilter.dll
NodeOutputFilter.pdb

cdx_v11.asmx
Global.asax
test_v10.asmx
web.config