Washington State Department of Ecology

Demonstrated Network Node Configuration

November 17, 2003

Version: 1.0

Prepared By

WINDSOR
SOLUTIONS, INC.

4000 Kruse Way Place

Building 2, Suite 160

Lake Oswego, OR 97035

(503) 675-7833

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

The National Environmental Information Exchange Network (Network) is an innovative approach for the exchange of environmental data between EPA, States and other partner organizations. The Network will promote access to and exchange of quality environmental data while reducing reporting burden and increasing efficiency of data exchanges. The Network is expected to become the preferred method for routine inter-governmental transfers of environmental data. The goal of the Network is to apply the Internet-based standards and technologies that have transformed information interchange generally to the specific needs of organizations engaged in environmental protection.

In order to participate in information exchange through the Network, each partner will establish an entry point or network node (Node). A Node is a simple environmental information Web service that initiates requests for information, processes authorized queries, and sends and receives the requested information in a standardized XML format. A Web service is software that exposes an organization's application functionality through the Internet using standards-based technologies that can be accessed by other entities independently of either party's technical environment.

This document describes the architecture of the Node developed by the State of Washington Department of Ecology (Ecology) in partnership with Windsor Solutions, Inc. (Windsor). The Node was implemented in August 2003, and was the first Node to support a production Facility Registry System (FRS) data flow with EPA via the Central Data Exchange Node. The Node was built in conformance with the *Network Node Functional Specification Version 1.0* and *Network Exchange Protocol Version 1.0*. The Node was recently updated to accommodate the changes detailed by the later *Network Node Functional Specification Version 1.1* and *Network Exchange Protocol Version 1.1* documents, and was successfully deployed and tested with these modifications installed.

The Ecology Node utilizes Microsoft's .NET technology and provides a robust and cost-effective Node implementation. It has been designed and built to work without any modification, in a Microsoft server environment, using SQL Server as the RDBMS. However, the Node's component-based architecture also allows for platform flexibility, and this Node is currently being customized for use against an Oracle 9i RDBMS for the Kansas Department of Health and Environment.

Implementation of the Node was designed to be as simple as possible, with the intention that additional Network partners should be able to implement a similar configuration with only minimal effort. The key implementation steps are:

1. Establish Node deployment environment
2. Create and Configure the Node Database
3. Configure the Node Database Account
4. Deploy the Node Executable
5. Configure the Node Application

This document describes the Ecology Node configuration and includes an implementation guide that can be used by a Network partner as the basis for the deployment of their own Node. The necessary application source code, executables and deployment files are provided with this document and are intended to facilitate the rapid and effective deployment of a working Node in most environments.

This goal of this document is to assist Network partners by reducing the effort, time, and cost associated with Node implementation.

Introduction

The National Environmental Information Exchange Network (Network) is an innovative approach for the exchange of environmental data between EPA, States and other partner organizations. The Network will promote access to and exchange of quality environmental data while reducing reporting burden and increasing efficiency of data exchanges. The Network is expected to become the preferred method for routine inter-governmental transfers of environmental data. The goal of the Network is to apply the standards and technologies that have transformed information interchange over the Internet to the specific needs of organizations engaged in environmental protection.

In order to participate in information exchange through the Network, each partner will establish an entry point or network node (Node). A Node is a simple environmental information Web service that initiates requests for information, processes authorized queries, and sends and receives the requested information in a standardized XML format. A Web service is software that exposes an organization's application functionality through the Internet using standards-based technologies that can be accessed by other entities independently of either party's technical environment.

For additional information concerning the principles and goals behind the Network implementation and the role of the Nodes in meeting these goals, the reader is directed to the *Implementation Plan for the National Environmental Information Exchange Network Version 1.0* published by the Network Steering Board.

The State of Washington Department of Ecology (Ecology) engaged Windsor Solutions, Inc (Windsor) to assist the agency with the analysis, design and construction of a Node. Development of the Node began in June 2003 and the production Node was implemented in August 2003. The implemented Node has fully automated the exchange of data between the Ecology Facility Site system and the EPA Facility Registry System via the EPA Central Data Exchange node (CDX). The Ecology Node is fully compliant with the following Network requirements:

- Network Exchange Protocol Version 1.1
- Network Node Functional Specification Version 1.1
- Network Security Guidelines and Recommendations

Before beginning development of the Node, Ecology considered a variety of alternative approaches to Node implementation, including turnkey products and demonstrated node configurations from other States. These alternatives were considered in light of Ecology's existing technical infrastructure investments and future information exchange requirements. Based on this analysis, Ecology chose to develop a custom Web services Node using the latest Microsoft technologies, specifically the .Net Framework version 1.1, and Visual Studio .Net 2003. The advanced Web services and XML handling capabilities of these Microsoft technologies were used to satisfy the Network specification and protocol requirements.

In addition to the development of the core Node functionality, an initial data flow was also developed to meet the needs for exchange of regulated facility information between Ecology's Facility Site system and the EPA Facility Registry System. Following two months of development and internal testing, the Ecology Node development team, together with Windsor, worked closely with EPA's CDX team to validate the capabilities of the Ecology Node and the CDX Node for management of this data flow. This joint effort resulted in a working flow of facility data between the two Network partners, and identified some necessary changes to the Network Node specification and protocol documents that were incorporated in to the *Network Node Functional Specification Version 1.1* and *Network Exchange Protocol Version 1.1*.

Ecology is currently actively pursuing the agency's goal to employ the Network concepts and Node capabilities to reengineer a number of types of electronic information exchange with other state and local government, and other private organizations.

Together with the States of Alaska, Idaho, and Oregon, Ecology is undertaking the development of an information interchange mechanism that seeks to facilitate the aggregation of and access to a comprehensive source of data related to ambient water quality in the Pacific Northwest. This mechanism is being developed jointly by the State participants using funds allocated from the EPA Network Challenge Grant and will be known as the Pacific Northwest Water Quality Data Exchange (Exchange). The Exchange will embody the Network concepts and will eventually be made available for sharing of water quality information throughout the Pacific Northwest region. Ecology is presently undertaking the design and development of the Web services needed to support the Exchange data flow and these are expected to be used for pilot data transfers early in 2004.

Additionally, Ecology is also engaged in a separate pilot project with the State of Oregon Department of Environmental Quality (ODEQ) to investigate the feasibility of the exchange of data concerning inter-state shipments of hazardous waste between regulated RCRA generator and treatment, storage and disposal facilities (TSDF). For example, with this pilot, the two agencies plan to share waste shipment information held in their RCRA program systems, such that a given generator's report of waste shipment to a TSDF in the other State can be compared directly to the records held in the opposing agency system for that TSDF.

Document Organization

The purpose of this document is to provide a reference for Network partners to use when establishing their own nodes, this document provides configuration instructions, recommended hardware and software requirements, setup and configuration activities, and Node administration techniques.

This document is intended to be used in conjunction with the *Network Exchange Protocol Version 1.1*, *Network Node Functional Specification Version 1.1*, and *Network Security Guidelines and Recommendations* documents, which provide the rules, guidelines and detailed description of the Network Node requirements. Where appropriate, references are made throughout this document to the specification and guideline documents rather than repeating these sections in this document. All documents related to the Network initiative can be accessed at the Exchange Network's site (http://www.exchangenetwork.net).

The remainder of this document has been organized as follows:

Functional Overview	provides a high-level description of the core components of the Ecology Node implementation.
Technical Architecture	describes the physical and logical structure of the Node application itself.
Installation Guide	outlines the system requirements and major steps needed to implement the Node.
Node Administration	describes the functionality of the Node administration tool provided with the Ecology Node.
Node Security	describes the security model used by the Ecology Node.
Establishing Data Flows	outlines the steps needed to implement a new data flow using the Ecology Node.

Functional Overview

The following figure illustrates the fundamental Node components and the interactions between these components. Each component is further described below.

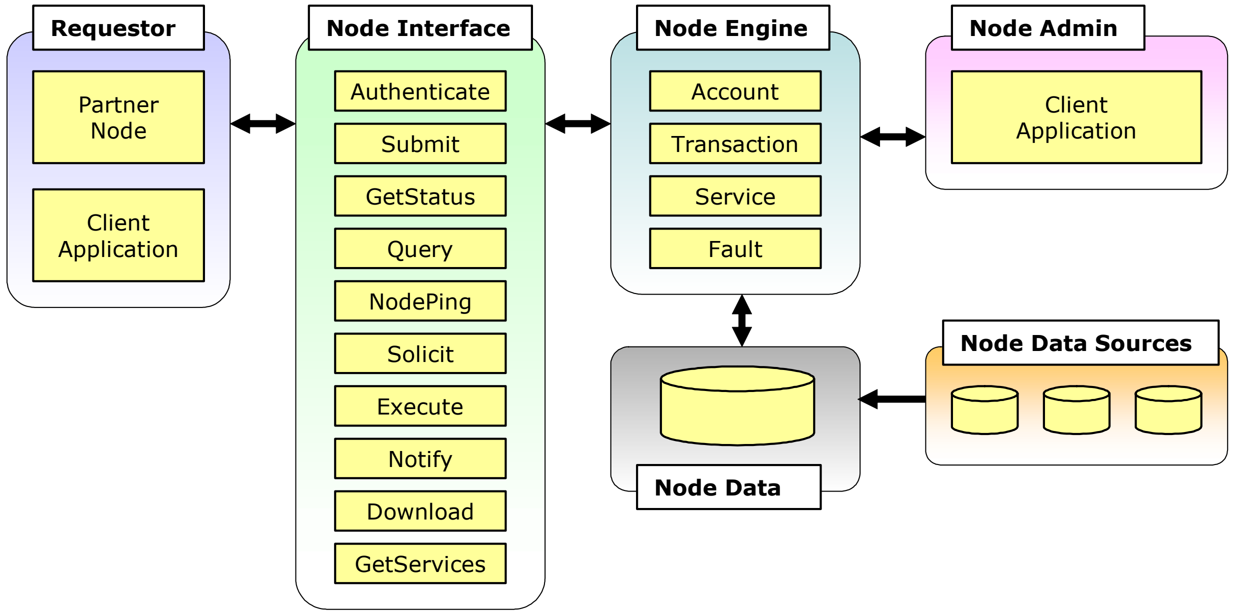


Figure 1: Node Overview

Requestor

The Web service functionality provided by the Ecology Node may be invoked in a number of ways by a requestor entity. For example, for the FRS data flow, the partner EPA CDX node issues requests to the Ecology Node on a regularly scheduled basis to collect updated information about regulated facilities. In the case of the pilot that Ecology is developing with ODEQ to exchange hazardous waste shipment information, the ODEQ hazardous waste program application system will issue a request to the Ecology Node to obtain waste shipment information for a given RCRA generator or TSDF.

Node Interface

The *Network Node Functional Specification Document Version 1.1* requires each network node to utilize a “remote procedure call” or RPC methodology to process incoming requests for information against the partner’s data sources. The specification calls for each network node interface to include the following ten standard Web methods:

- Authenticate
- Submit
- GetStatus
- Query
- NodePing
- Solicit

- Execute (optional)
- Notify
- Download
- GetServices

The function of each of these methods is described in detail in the specification document. Besides providing operational information about the partner node and ensuring that the appropriate security controls are applied, these methods serve to accept and pass on incoming requests from a requestor to either receive or supply information. These requests may be generated from another partner node or from a custom client application. For example, a requestor may ask a node service for a list of regulated facilities located in a given county, by invoking the "Query" method and providing the service name and the required parameters in a predetermined order.

Node Engine

The Node interface will pass on an incoming request to the Node Engine. The Node Engine parses the incoming request and then invokes the appropriate network service to respond to the request. Using the previous example, the Facility Registry System service or request processor will be invoked and asked to execute the "GetFacilitiesByCounty" method. The requested method will then interrogate the appropriate partner data source, in this case, data from the Ecology Facility Site database.

The flow-specific request processor will then transform the results from the execution of the query into an XML document using the appropriate schema definition and will return a reference to this document to the Node Engine. The Node Engine in turn then returns this document to the requestor through the node interface. The node is characterized as using an RPC methodology because neither the type or number of parameters passed to the node Query method, nor the values returned are described by that method. Instead, these values are simply passed on as an array.

New flow processors can be built and implemented in the Node Engine as required, for example, to support the flow of information to the EPA RCRAInfo national system.

Node Data Sources

When invoked by the Node Engine, the relevant request processor will execute one or more queries against a source of the relevant data. Rather than execute these queries directly against the agency program databases, the Ecology Node architecture was designed to include a single specific database for Node Data. This Node Data source is refreshed regularly with data from the program databases. The isolation of the true data source from the Node request processing functions also improves efficiency since the Node Data source is modeled to better support the Node queries than the program data source itself may have been.

Node Administration Client

The implemented Ecology Node includes a client-based administration tool, the NodeAdmin utility, which provides the user with remote administration capabilities through an easy to use Windows-based interface. This client utility allows the user to manage user accounts and the services available to each user, as well as providing general Node operational support functions.

The functionality of the NodeAdmin utility is discussed in detail in the *Node Administration* section of this document.

Node Testing Client

Alongside the development of the Ecology Node, Windsor has developed a comprehensive testing tool based on the *Network Node Functional Specification Version 1.1*, and *Network Security Guidelines and Recommendations* documents that will allow a user to interface with any available Network node. This powerful application is intended to assist Network partners in testing and evaluating Network nodes, by:

- Helping partners to understand the purpose and workings of the Network via a real-world demonstration,
- Assisting partners in testing the functionality of their Node while it is being developed and/or deployed, and
- Providing partners with a temporary, semi-automated alternative to a fully functioning Node that will allow a partner to submit XML files to another partner Node, and/or solicit, query or download XML data (or other such payloads).

This testing tool, the NodeWinClient utility, is provided for free distribution to Network partners to assist them in their Node testing. The latest version of the NodeWinClient utility can be downloaded from http://www.windsorsolutions.biz/nodeclient/.

Technical Architecture

Physical Topology

The following diagrams illustrate two possible physical configurations for the Node.

The typical configuration assumes that the node components and node data source would reside in a secure DMZ environment, isolated from the outside world and the internal agency environment by appropriate firewall structures.

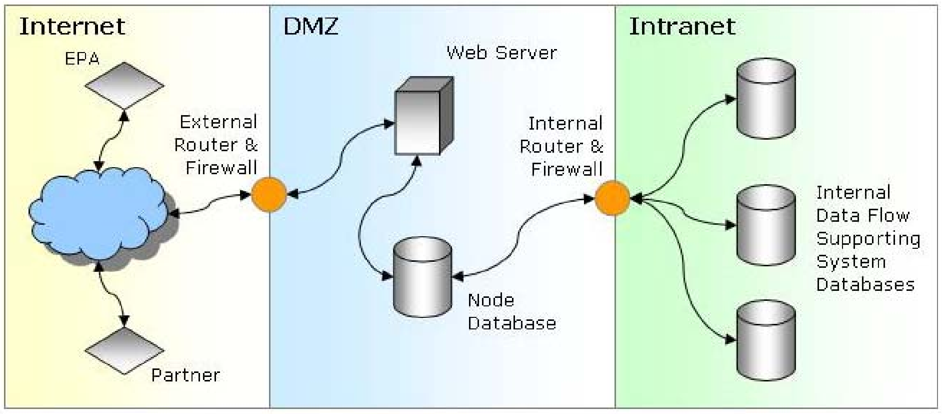

Internet
EPA
External Router & Firewall
Partner
DMZ
Web Server
Node Database
Internal Router & Firewall
Intranet
Internal Data Flow Supporting System Databases

Figure 2: Typical Configuration

The Ecology Node, however, adopted a slightly less common configuration owing to the agency's established Web server infrastructure. In this case, the Node was configured behind an HTTP Proxy Server (Fortress) and as such had to accommodate certain specific network requirements.

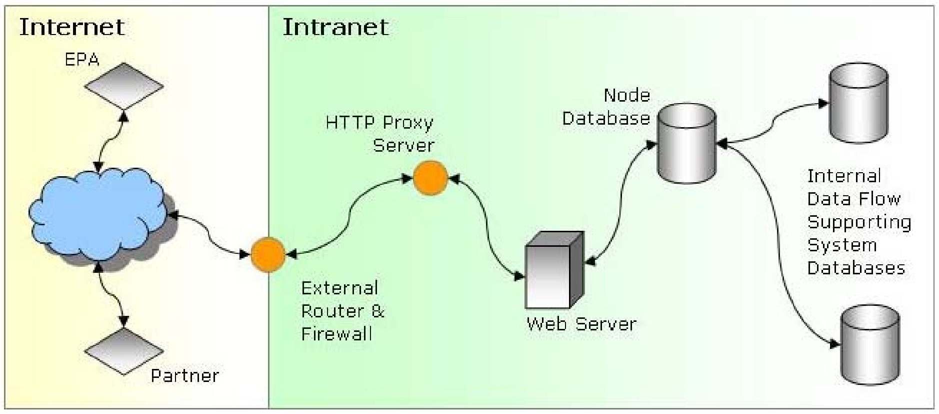

Internet
EPA
Partner
Intranet
HTTP Proxy Server
External Router & Firewall
Node Database
Web Server
Internal Data Flow Supporting System Databases

Figure 3: HTTP Proxy Based Configuration (Ecology-specific)

The Ecology Node was developed to be flexible enough to accommodate other common network environments. The Node itself has been tested with EPA's CDX node and shown to support both of the two physical network architectures illustrated above.

Logical Architecture

Like most enterprise-level systems, the Node application was designed in logical layers (these layers may, or may not directly correspond to the physical architecture). This approach improves the Node's deployment flexibility and its robustness, and simplifies ongoing maintenance.

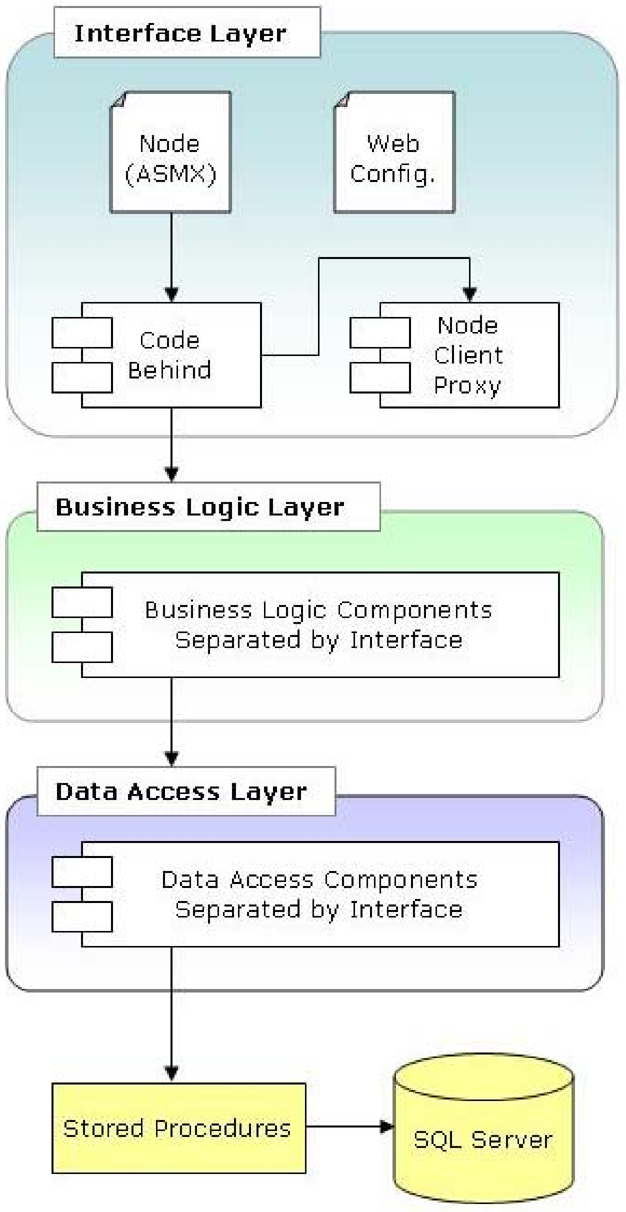


Figure 4: Node Logical Layers

Interface Layer

The interface layer includes the Node's external services and Web methods as defined in the *Network Node Functional Specification Version 1.1* and *Network Exchange Protocol Version 1.1*. These services are classified into four major abstract interfaces. In the Ecology Node implementation, these interfaces are represented by one object (node.asmx) and correspond to the components of the business logic layer.

The reader is directed to the relevant documents for a complete description of each method and interface.

Business Logic Layer

The business logic layer encapsulates all of the business logic required for the operation of the Node application. The business logic layer is composed of .NET assemblies developed using C#. The component classes in the business logic layer were developed with an emphasis on code reusability through the use of Object Oriented Programming.

The fully independent business logic layer components are capable of supporting the Node Web methods located in the interface layer as well as any other application implementing these components, for example, the Node Administration application discussed later in this document, which utilizes these same components to interact with the Node.

The business logic layer components are described in detail in the *Object Model* discussion later in this section.

Data Access Layer

The Node data access layer provides all data services to the business logic layer. This layer is implemented as a single .Net assembly using the Microsoft Data Access Application Block (SQLHelper).

SQLHelper is a .NET component that contains optimized data access code to call stored procedures and issue SQL text commands against a SQL Server database. It returns SqlDataReader, DataSet, and XmlReader objects. The Node utilizes SQLHelper as a compiled assembly to reduce the amount of custom code, decrease testing, and increase maintainability.

The data access layer is responsible for management and processing of all interactions between the business logic layer and the Node database.

Object Model

The following figure illustrates the component classes included in the business logic layer.

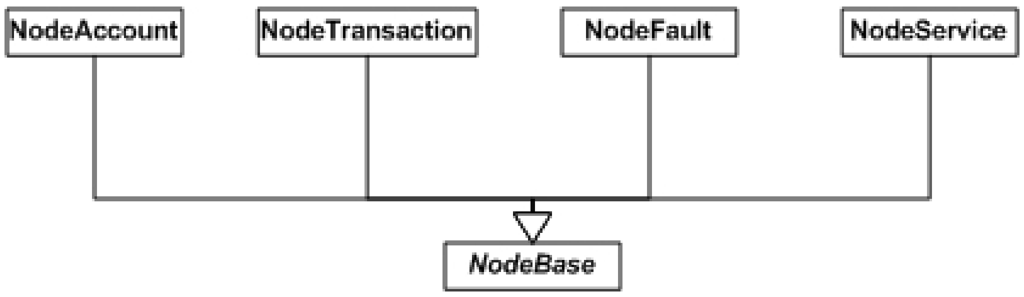
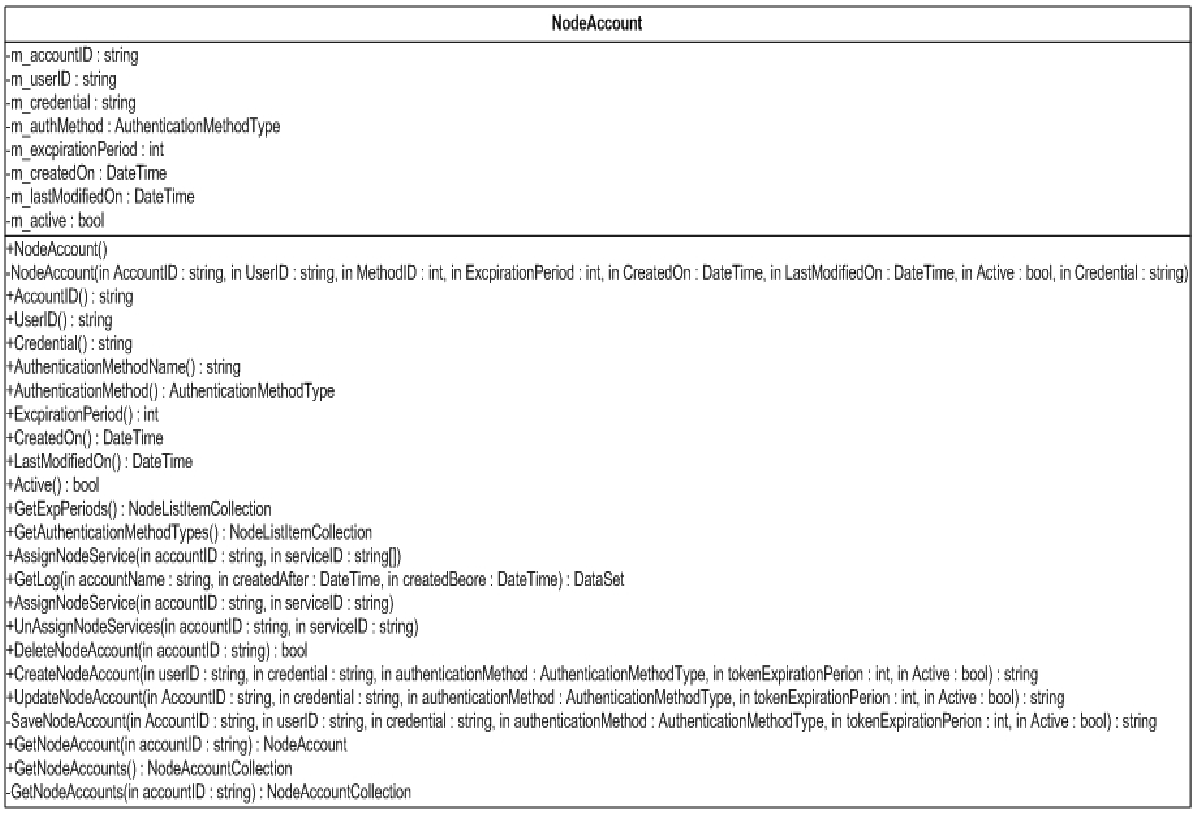


Figure 5: Node BLL Object Diagram

NodeAccount Class

The NodeAccount class represents the Account entity in the Node database and is responsible for all Node account management. This object interacts with both the Node service as well as the NodeAdmin utility.

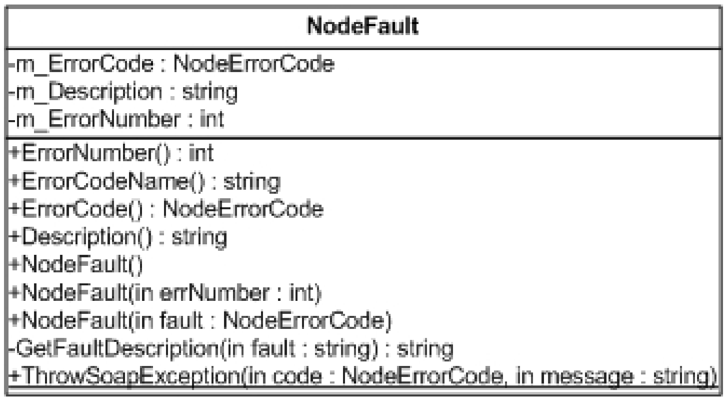
The structure of this class is as follows:



```
NodeAccount
-m_accountID : string
-m_userID : string
-m_credential : string
-m_authMethod : AuthenticationMethodType
-m_excpirationPeriod : int
-m_createdOn : DateTime
-m_lastModifiedOn : DateTime
-m_active : bool
+NodeAccount()
-NodeAccount(in AccountID : string, in UserID : string, in MethodID : int, in ExcpirationPeriod : int, in CreatedOn : DateTime, in LastModifiedOn : DateTime, in Active : bool, in Credential : string)
+AccountID() : string
+UserID() : string
+Credential() : string
+AuthenticationMethodName() : string
+AuthenticationMethod() : AuthenticationMethodType
+ExcpirationPeriod() : int
+CreatedOn() : DateTime
+LastModifiedOn() : DateTime
+Active() : bool
+GetExpPeriods() : NodeListItemCollection
+GetAuthenticationMethodTypes() : NodeListItemCollection
+AssignNodeService(in accountID : string, in serviceID : string[])
+GetLog(in accountName : string, in createdAfter : DateTime, in createdBeore : DateTime) : DataSet
+AssignNodeService(in accountID : string, in serviceID : string)
+UnAssignNodeServices(in accountID : string, in serviceID : string)
+DeleteNodeAccount(in accountID : string) : bool
+CreateNodeAccount(in userID : string, in credential : string, in authenticationMethod : AuthenticationMethodType, in tokenExpirationPerion : int, in Active : bool) : string
+UpdateNodeAccount(in AccountID : string, in credential : string, in authenticationMethod : AuthenticationMethodType, in tokenExpirationPerion : int, in Active : bool) : string
-SaveNodeAccount(in AccountID : string, in userID : string, in credential : string, in authenticationMethod : AuthenticationMethodType, in tokenExpirationPerion : int, in Active : bool) : string
+GetNodeAccount(in accountID : string) : NodeAccount
+GetNodeAccounts() : NodeAccountCollection
-GetNodeAccounts(in accountID : string) : NodeAccountCollection
```

NodeFault Class

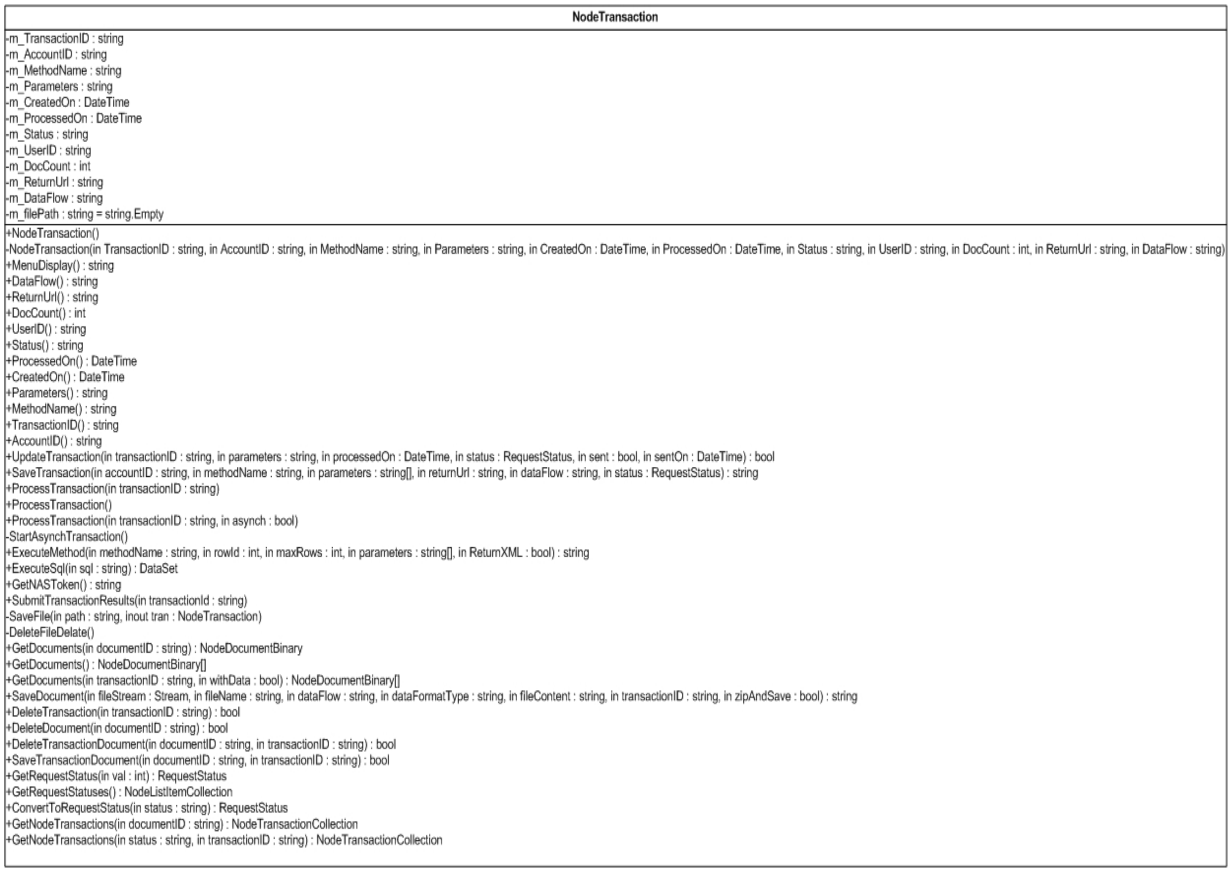
The NodeFault class represents the SOAP Fault as specified by the *Network Node Functional Specification Version 1.1*. Its purpose is to provide a consistent and reliable way of communicating all Node errors back to the requestor.



```
NodeFault
-m_ErrorCode : NodeErrorCode
-m_Description : string
-m_ErrorNumber : int
+ErrorNumber() : int
+ErrorCodeName() : string
+ErrorCode() : NodeErrorCode
+Description() : string
+NodeFault()
+NodeFault(in errNumber : int)
+NodeFault(in fault : NodeErrorCode)
-GetFaultDescription(in fault : string) : string
+ThrowSoapException(in code : NodeErrorCode, in message : string)
```

NodeTransaction Class

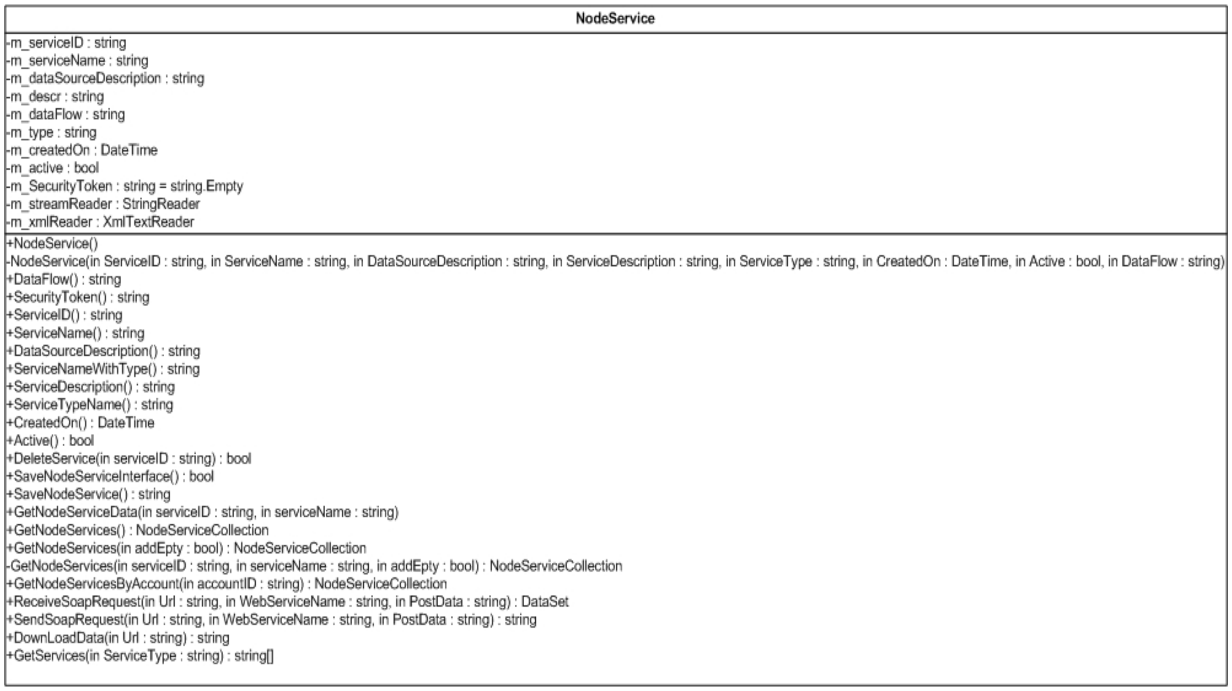
The NodeTransaction class represents the Node transaction entity and is responsible for managing all transaction requests. NodeTransaction also interacts directly with the individual dataflow processing object through the means of reflection. More detailed information about the processing of data flow requests as well as the creation of new data flows and their deployment through the Node can be found in the *Establishing Data Flows* section of this document.



```
NodeTransaction
-m_TransactionID : string
-m_AccountID : string
-m_MethodName : string
-m_Parameters : string
-m_CreatedOn : DateTime
-m_ProcessedOn : DateTime
-m_Status : string
-m_UserID : string
-m_DocCount : int
-m_ReturnUrl : string
-m_DataFlow : string
-m_filePath : string = string.Empty
+NodeTransaction()
-NodeTransaction(in TransactionID : string, in AccountID : string, in MethodName : string, in Parameters : string, in CreatedOn : DateTime, in ProcessedOn : DateTime, in Status : string, in UserID : string, in DocCount : int, in ReturnUrl : string, in DataFlow : string)
+MenuDisplay() : string
+DataFlow() : string
+ReturnUrl() : string
+DocCount() : int
+UserID() : string
+Status() : string
+ProcessedOn() : DateTime
+CreatedOn() : DateTime
+Parameters() : string
+MethodName() : string
+TransactionID() : string
+AccountID() : string
+UpdateTransaction(in transactionID : string, in parameters : string, in processedOn : DateTime, in status : RequestStatus, in sent : bool, in sentOn : DateTime) : bool
+SaveTransaction(in accountID : string, in methodName : string, in parameters : string[], in returnUrl : string, in dataFlow : string, in status : RequestStatus) : string
+ProcessTransaction(in transactionID : string)
+ProcessTransaction()
+ProcessTransaction(in transactionID : string, in asynch : bool)
-StartAsynchTransaction()
+ExecuteMethod(in methodName : string, in rowId : int, in maxRows : int, in parameters : string[], in ReturnXML : bool) : string
+ExecuteSql(in sql : string) : DataSet
+GetNASToken() : string
+SubmitTransactionResults(in transactionId : string)
-SaveFile(in path : string, inout tran : NodeTransaction)
-DeleteFileDelate()
+GetDocuments(in documentID : string) : NodeDocumentBinary
+GetDocuments() : NodeDocumentBinary[]
+GetDocuments(in transactionID : string, in withData : bool) : NodeDocumentBinary[]
+SaveDocument(in fileStream : Stream, in fileName : string, in dataFlow : string, in dataFormatType : string, in fileContent : string, in transactionID : string, in zipAndSave : bool) : string
+DeleteTransaction(in transactionID : string) : bool
+DeleteDocument(in documentID : string) : bool
+DeleteTransactionDocument(in documentID : string, in transactionID : string) : bool
+SaveTransactionDocument(in documentID : string, in transactionID : string) : bool
+GetRequestStatus(in val : int) : RequestStatus
+GetRequestStatuses() : NodeListItemCollection
+ConvertToRequestStatus(in status : string) : RequestStatus
+GetNodeTransactions(in documentID : string) : NodeTransactionCollection
+GetNodeTransactions(in status : string, in transactionID : string) : NodeTransactionCollection
```

NodeService Class

The NodeService class represents the node service entity and is responsible for the management of all service related request. This class is closely coupled with the NodeTransaction class and together these classes are responsible for processing all queries and solicit requests.



```
NodeService
-m_serviceID : string
-m_serviceName : string
-m_dataSourceDescription : string
-m_descr : string
-m_dataFlow : string
-m_type : string
-m_createdOn : DateTime
-m_active : bool
-m_SecurityToken : string = string.Empty
-m_streamReader : StringReader
-m_xmlReader : XmlTextReader
+NodeService()
-NodeService(in ServiceID : string, in ServiceName : string, in DataSourceDescription : string, in ServiceDescription : string, in ServiceType : string, in CreatedOn : DateTime, in Active : bool, in DataFlow : string)
+DataFlow() : string
+SecurityToken() : string
+ServiceID() : string
+ServiceName() : string
+DataSourceDescription() : string
+ServiceNameWithType() : string
+ServiceDescription() : string
+ServiceTypeName() : string
+CreatedOn() : DateTime
+Active() : bool
+DeleteService(in serviceID : string) : bool
+SaveNodeServiceInterface() : bool
+SaveNodeService() : string
+GetNodeServiceData(in serviceID : string, in serviceName : string)
+GetNodeServices() : NodeServiceCollection
+GetNodeServices(in addEpty : bool) : NodeServiceCollection
-GetNodeServices(in serviceID : string, in serviceName : string, in addEpty : bool) : NodeServiceCollection
+GetNodeServicesByAccount(in accountID : string) : NodeServiceCollection
+ReceiveSoapRequest(in Url : string, in WebServiceName : string, in PostData : string) : DataSet
+SendSoapRequest(in Url : string, in WebServiceName : string, in PostData : string) : string
+DownLoadData(in Url : string) : string
+GetServices(in ServiceType : string) : string[]
```

Data Model

The following figure illustrates the data model used for the underlying database that supports the Node business logic layer operations discussed in the previous sections.

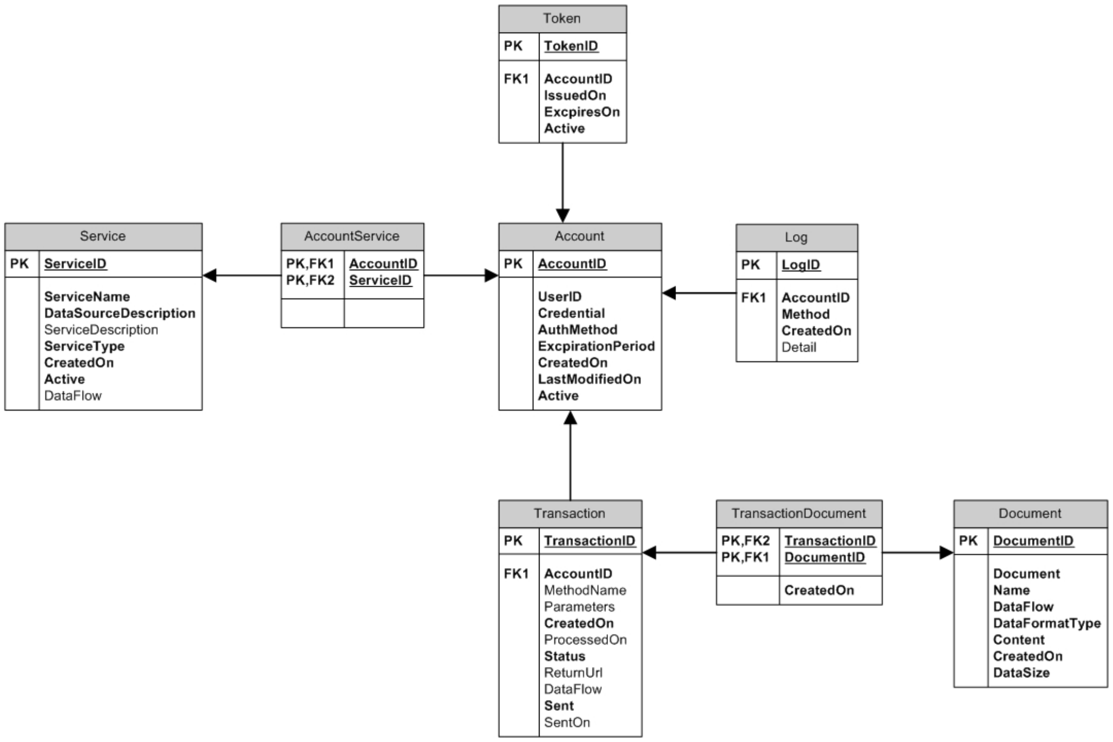


Figure 6: Node Data Model

Installation Guide

The accompanying zip file contains the setup files needed to install and administer the Node. All source code is available upon request from Debbie Stewart, tel: (360) 407 7048, e-mail: dste461@ecy.wa.gov.

System Requirements

The Node Web Services application has been design to be hosted from a Microsoft .NET platform. The following hardware and software configuration is recommended for this platform. In addition, it is recommended that the reader consult the appropriate product documentation for the most recent requirements and security configurations.

Server Software

Microsoft Server 2003, Enterprise Edition

The Node application was developed and tested on the enterprise version of 2003 Server. The application was successfully tested and deployed also in Windows 2000 environment. Please refer to the Microsoft server documentation for detailed operating system hardware requirements[1].

Microsoft Internet Information Server (IIS) 6.0

When deployed in a single machine configuration, the Node application will utilize the IIS installation on the hosting server. The Node application has also been successfully tested with an IIS 5.0 (Windows 2000 Server) installation.

Microsoft SQL Server 2000 (SP3)

The Node Web Services application, as well as the NodeAdmin utility, were developed against SQL Server 2000. The database schema and the Node stored procedures are also compatible with SQL 7.0 or MDAC 2000.

Additional Software

Microsoft .NET Framework 1.1 (v1.1.4322)

The Node application as well as the NodeAdmin utility was developed using the Microsoft .NET Framework 1.1.

Web Services Enhancements 1.0 (WSE)

To accommodate the Node Direct Internet Message Encapsulation (DIME) and SSL requirements the Node application was developed using Microsoft Web Services Enhancements, an add-on to the Microsoft .NET Framework providing the latest advanced Web services capabilities.

[1] http://www.microsoft.com/windowsserver2003/evaluation/sysreqs/default.mspx

Microsoft Data Access Components 2.7 (MDAC)

The Node application uses MDAC to communicate with the underlying Node database. The application server hosting the Node Engine component must be configured with this MDAC version.

Node Deployment

This section provides guidelines for deployment of the Ecology Node in a technical environment compatible with the hardware and software specifications outlined above.

The deployment process includes the following tasks:

1. Creation and configuration of Node database
2. Configuration of Node database accounts.
3. Deployment of Node executables.
4. Configuration of Node application.

Creation and configuration of Node database

Requirements:

SQL Server installation meeting minimal previously outlined hardware requirements

SQL Server Administrator rights

NodeUser and NodeAdmin[2] SQL Server user accounts associated to the Node database.

Installation Files:

node.sql script

The Node comes with a preconfigured Node database creation script. The node.sql script will create and configure the Node database in the default location specified by the SQL Server installation. Alternatively, the Node database can be first created in the desired location and the provided script used to create only the necessary tables and stored procedures.

The Node utilizes two named accounts to access the database. The NodeUser account is used by the Node clients accessing the Node data. The NodeAdmin account is used by the NodeAdmin application described later in this document to manage and administer the Node itself.

The provided script grants the necessary execute rights to the appropriate stored procedures to each of these users. The users themselves have no rights on the database tables.

For a further description of the Ecology Node security model, see the *Node Security* section of this document

[2] The NodeUser and NodeAdmin are used for presentation only. The actual user accounts can have different names but they have to be granted public access to the Node database.

Deployment of Node executables

Requirements:

IIS Server installation meeting minimal previously outlined hardware requirements

Local administrator rights

Installation Files:

node.msi package

The Microsoft Setup and Install (MSI) package will create the necessary directories, configure the IIS server and copy all necessary application executable files. The reader is directed to the detailed installation instructions included in the MSI package.

Configuration of Node application

Requirements:

Successfully deployed and tested Node database

IIS Node application folder containing Node executables

Installation Files:

Web.config file (will already exist in the Node application directory)

The .Net Framework provides an alternative to storing application configuration information in the machine registry. The Web.config file describes all of the parameters required to fully configure the Node application.

Key	**Purpose**
ConnectionString	Connection string to the SQL Server database for the Node application.
RequireSSL	True or False flag indicating whether the Node application should be requiring SSL communication. In some HTTP Proxy configurations the SSL certificate resides on the proxy and communication between the proxy and Node server is conducted over the standard HTTP protocol (80).
EnableExecuteMethod	True or False flag indicating whether the Node application should be responding to the Execute method. (Considering the security implications and the fact that the Node specification 1.1 designates this method as optional, the default configuration of the Node application has this method turned off.
MaxRowsReturn	Default number of maximum records returned by the Query method. The solicit method does not utilize this option and returns all data specified by the individual flows.
TempFolderPath	Location of the directory where dataflow objects will be serialized. Make sure that the final trailing slash is there. Node ASP user used by the .NET Framework must have Read/Write/Modify rights to this directory.
ExternalURL	Fully qualified URL for the Node. Used with the HTTP Proxy as a workaround for the Node Fault messages. (Proxy Specific)
UseHTTPProxy	True or False flag used to turn the HTTP Proxy specific options. Turn this on

Key	Purpose
	if the Node Web server resides behind a proxy. (Proxy Specific)
CDXHostIP	Because the reversed proxy prevents Node from seeing the actual IP address of the requestor, you must enter a static IP address to allow the CDX centralized token authentication to work properly. (Proxy Specific)
CDXAccount	Account name for this node at CDX.
CDXCredentials	Credential for the above account at CDX
NASUrl	URL of the NAAS Authentication WSDL file.
CDXNode	URL to the CDX Node.
FlowTitle	The Node uses reflection to identify specific flow processors (DLLs). To allow maximum flexibility, the Node allows for loosely coupled assemblies to perform the actual data flow specific requests handing. Please see the *Establishing Data Flows* section of this document for further information.

The Web.config file also contains a number of system wide configuration parameters. The following excerpt from the Web.config file illustrates the recommended production Node settings. During development and testing, the individual settings can be adjusted to allow for more debugging, browser-based testing and more verbose logging. The reader is directed to the Microsoft .Net Framework specifications for details on modifications to the Web.config file.

```
<system.web>
      <webServices>
            <soapExtensionTypes>
                  <add
                  type="Microsoft.Web.Services.WebServicesExtension,
                        Microsoft.Web.Services,
                  Version=1.0.0.0,
                        Culture=neutral,
                  PublicKeyToken=31bf3856ad364e35"
                  priority="1"
                  group="0"/>
            </soapExtensionTypes>
             <protocols>
                  <clear/>
                  <add name="HttpSoap"/>
                  <add name="Documentation"/>
            </protocols>
      </webServices>
      <compilation defaultLanguage="c#" debug="false"/>
      <customErrors mode="Off"/>
      <authentication mode="Windows"/>
      <authorization>
            <allow users="*"/>
      </authorization>
      <trace enabled="false"
```

```
                requestLimit="10"
                pageOutput="false"
                traceMode="SortByTime"
                localOnly="true"/>
     <sessionState mode="Off"/>
     <globalization
          requestEncoding="utf-8"
                responseEncoding="utf-8" />
</system.web>
```

Node Administration

The Node Administration utility (NodeAdmin) is a Windows Forms application based on the .NET Framework. NodeAdmin allows remote administration of the Node application through a series of windows described in this section.

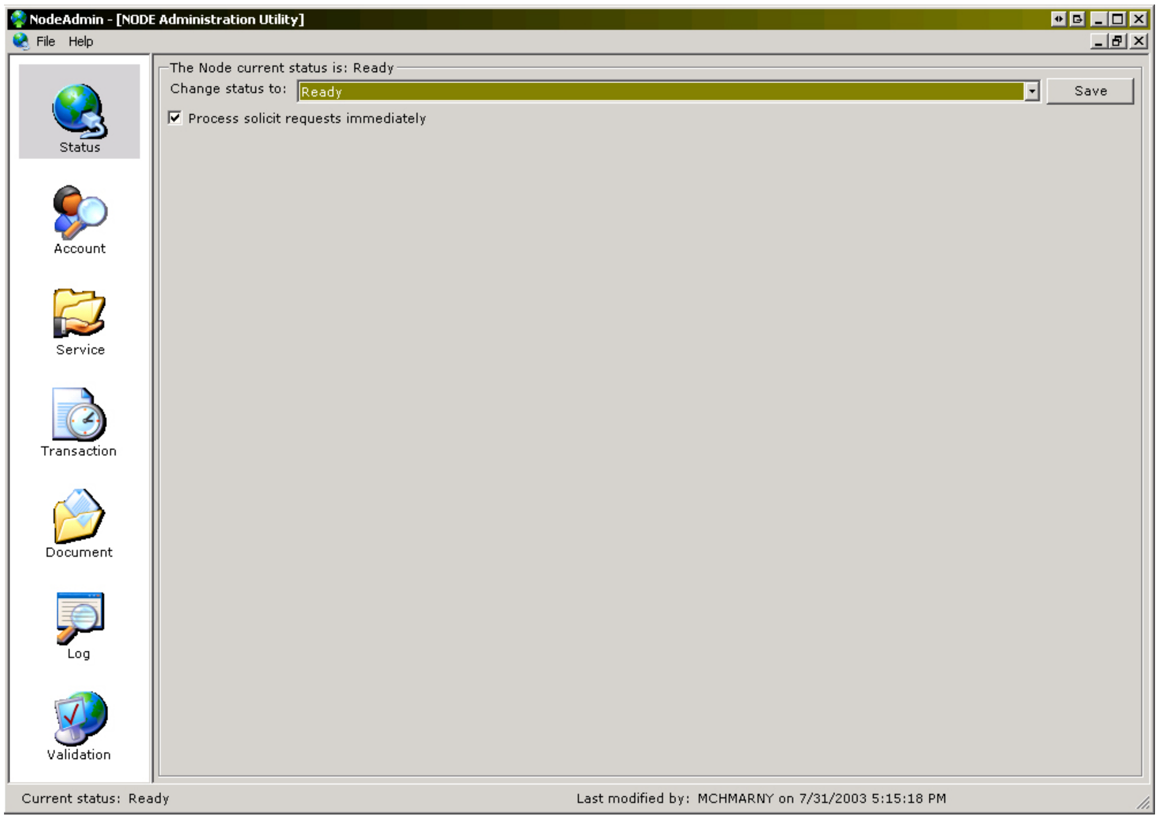
Requirements

The *NodeAdmin* utility has been design to allow a more user-friendly approach to managing the Node. The following hardware and software configuration is recommended for any workstation using the *NodeAdmin* utility.

- Windows XP/2000
- Microsoft .NET Framework 1.1 (v1.1.4322)
- Appropriate configuration file with access to the Node database

Manage Node Status

The Node Status window allows the Node Administrator (person) to modify the current status of the Node.



The Node current status is: Ready

Change status to: Ready

☑ Process solicit requests immediately

Current status: Ready

Last modified by: MCHMARNY on 7/31/2003 5:15:18 PM

The dropdown list of predefined status types includes:

Ready	the service is up and running

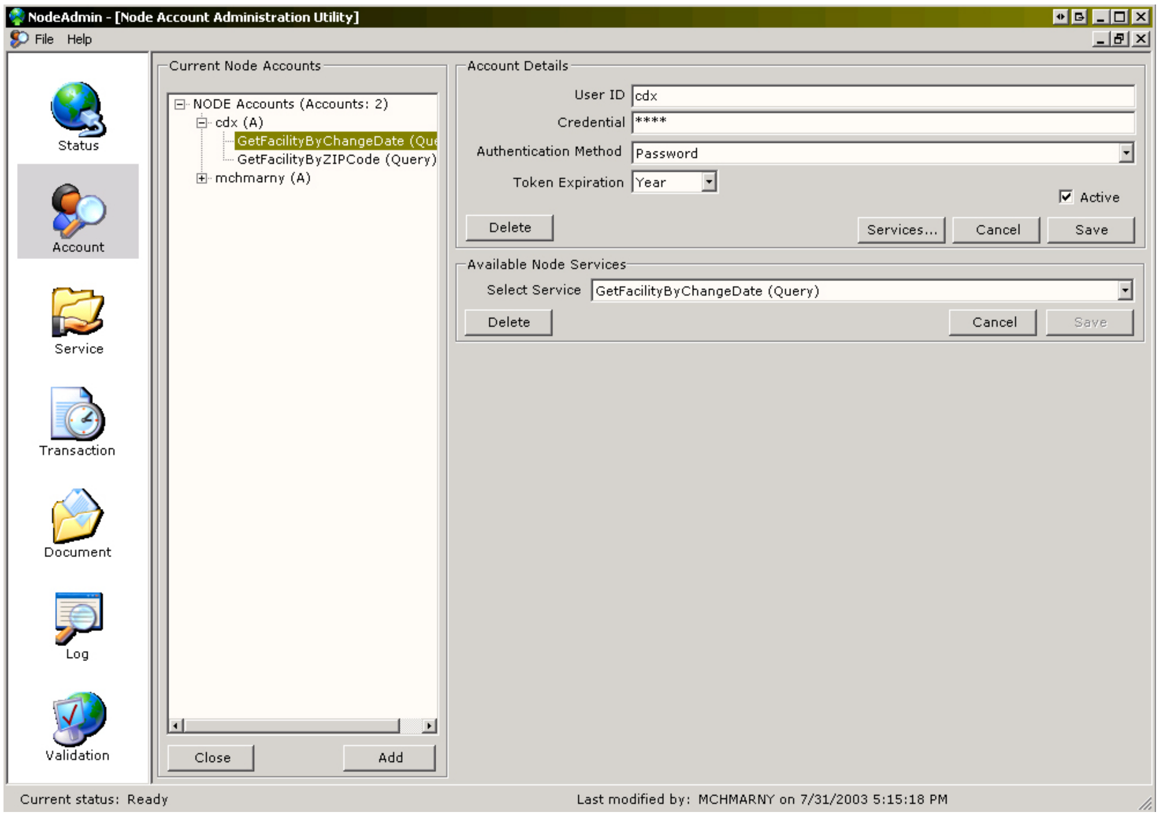
Busy	the service is heavily loaded, please call back later
Unavailable	the service is currently unavailable

Modifying the status will immediately impact the status of the Node.

Checking the "Process solicit requests immediately" checkbox on will set the Node to process the solicited queries in background (low priority process) immediately after the request is submitted. Checking this option off will require the Node Administrator to trigger these requests manually.

Manage Node Accounts

The Node Account window provides a three-pane interface to enable management of Node accounts.



NODE Accounts (Accounts: 2)
cdx (A)
GetFacilityByChangeDate (Qu
GetFacilityByZIPCode (Query)
mchmarny (A)

Account Details

User ID: cdx
Credential: ****
Authentication Method: Password
Token Expiration: Year
☑ Active

Available Node Services

Select Service: GetFacilityByChangeDate (Query)

Current status: Ready

Last modified by: MCHMARNY on 7/31/2003 5:15:18 PM

The tree view to left of the window presents a list of accounts currently configured in the Node database. The account status is indicated by a letter corresponding either to active (A) or inactive (I).

Clicking on an individual account activates its details located in the right-top pane, allowing the Node administrator to edit the account details and to view the particular services associated with this account.

The right- lower pane contains the list of services which can be associated with each account.

Authentication Method

Currently the Node supports only the password authentication method. Future releases will support other means of authentication (PKI, Digest, Certificate, SAML etc.).

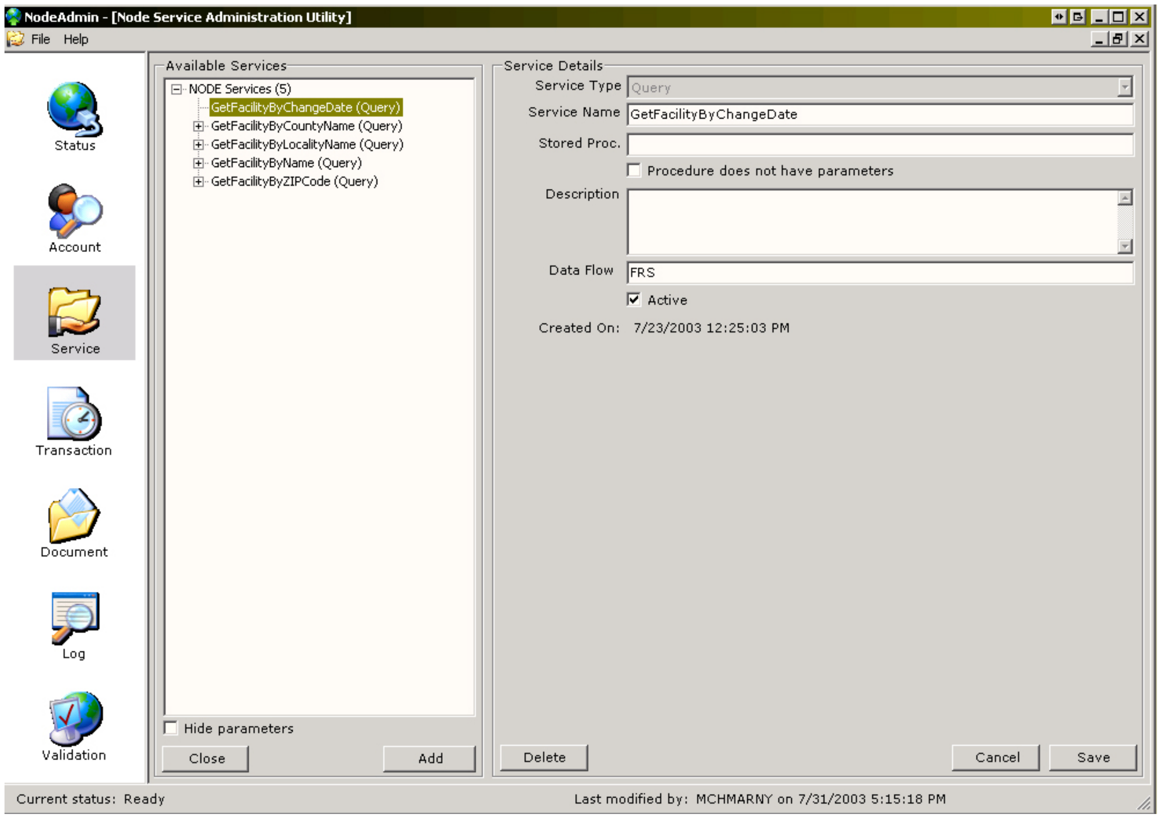
Token Expiration

The token expiration menu allows the Node administrator to specify the period after which each token will expire. Each token is validated prior to processing the user request, thus long running jobs exceeding the expiration period will continue to be processed.

Manage Node Services

The Node Service administration window includes the core functionality of the NodeAdmin utility. It allows management of the data flows configured within the Node database.

Note that prior to adding a new service within the NodeAdmin utility, that service has to be developed and configured as the specific information about that service will be obtained from the service WSDL file.



NODE Services (5)
- GetFacilityByChangeDate (Query)
- GetFacilityByCountyName (Query)
- GetFacilityByLocalityName (Query)
- GetFacilityByName (Query)
- GetFacilityByZIPCode (Query)

Service Details

Service Type: Query
Service Name: GetFacilityByChangeDate
Stored Proc.:
Procedure does not have parameters
Description:
Data Flow: FRS
[✓] Active
Created On: 7/23/2003 12:25:03 PM

Current status: Ready
Last modified by: MCHMARNY on 7/31/2003 5:15:18 PM

The tree on the left side of the screen lists the currently configured services supported by the Node. Clicking on any of the nodes of this tree will activate the Service detail pane showing the details of the Service

The Ecology Node currently supports three main types of Web Service:

Query

This type of service utilizes the RPC method. All results of this query are in a string format and correspond to predefined queries in SQL database. When using a simple stored procedure query, the parameters of that stored procedure will automatically be queried and saved on the Node database as its parameters.

Interface

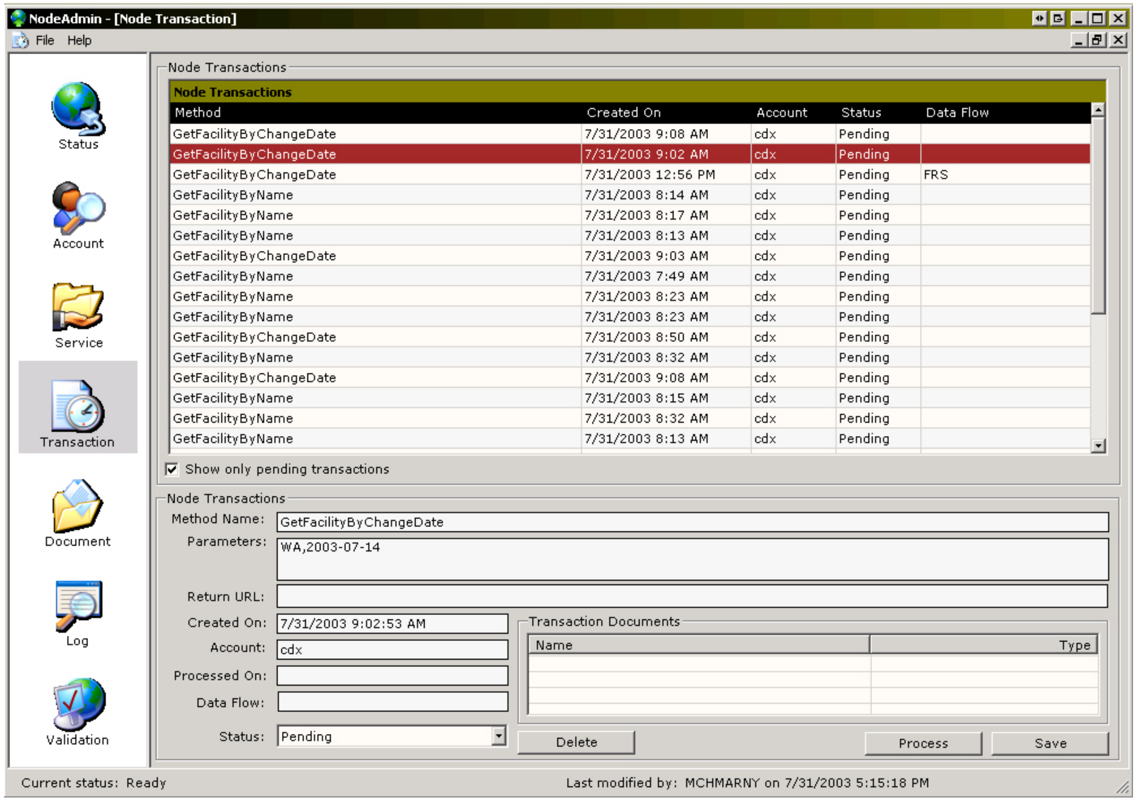
This type of service utilizes the document/literal method. All results of this query are strongly typed and represent a specific interface (an asmx file) which is described by its own WSDL file.

Execute

Due to potential security issues related to allowing users to execute dynamic SQL commands against the partner databases this method type has been disabled by default. This method can be enabled by modifying the AllowExecute parameter in the Node *Web.config* file.

Manage Node Transactions

The Node Transaction window tracks each transaction (both incoming as well as outgoing). A Node Transaction represents a specific request submitted through the Node interface.



Node Transactions

Method	Created On	Account	Status	Data Flow
GetFacilityByChangeDate	7/31/2003 9:08 AM	cdx	Pending	
GetFacilityByChangeDate	7/31/2003 9:02 AM	cdx	Pending	
GetFacilityByChangeDate	7/31/2003 12:56 PM	cdx	Pending	FRS
GetFacilityByName	7/31/2003 8:14 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:17 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:13 AM	cdx	Pending	
GetFacilityByChangeDate	7/31/2003 9:03 AM	cdx	Pending	
GetFacilityByName	7/31/2003 7:49 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:23 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:23 AM	cdx	Pending	
GetFacilityByChangeDate	7/31/2003 8:50 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:32 AM	cdx	Pending	
GetFacilityByChangeDate	7/31/2003 9:08 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:15 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:32 AM	cdx	Pending	
GetFacilityByName	7/31/2003 8:13 AM	cdx	Pending	

☑ Show only pending transactions

Node Transactions

Method Name: GetFacilityByChangeDate
Parameters: WA,2003-07-14
Return URL:
Created On: 7/31/2003 9:02:53 AM
Account: cdx
Processed On:
Data Flow:
Status: Pending

Transaction Documents

Name	Type

Current status: Ready   Last modified by: MCHMARNY on 7/31/2003 5:15:18 PM

By selecting the specific transaction from the list at the top of the screen, the Node Administrator can view and change the status of each transaction.

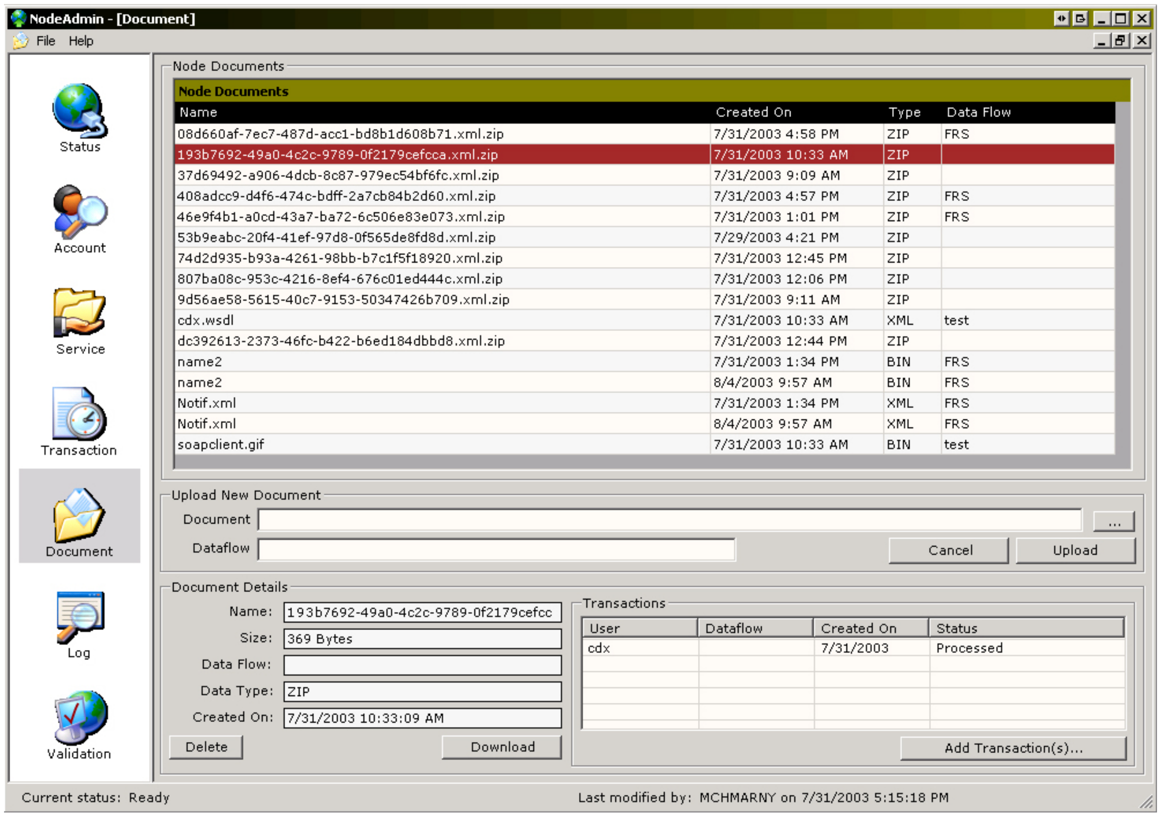
In addition, the documents associated with this transaction are listed in the Transaction Documents pane in the bottom right corner.

The transaction detail screen provides a manual triggering mechanism should the Process Solicit Requests Immediately option be checked off on the Status screen.

WINDSOR SOLUTIONS, INC.

Manage Node Documents

The Node Document window allows the Administrator to track each document. A Node Document represents a specific document either submitted through the node interface or generated by one of the Node data flows.



Node Documents

Name	Created On	Type	Data Flow
08d660af-7ec7-487d-acc1-bd8b1d608b71.xml.zip	7/31/2003 4:58 PM	ZIP	FRS
193b7692-49a0-4c2c-9789-0f2179cefcca.xml.zip	7/31/2003 10:33 AM	ZIP	
37d69492-a906-4dcb-8c87-979ec54bf6fc.xml.zip	7/31/2003 9:09 AM	ZIP	
408adcc9-d4f6-474c-bdff-2a7cb84b2d60.xml.zip	7/31/2003 4:57 PM	ZIP	FRS
46e9f4b1-a0cd-43a7-ba72-6c506e83e073.xml.zip	7/31/2003 1:01 PM	ZIP	FRS
53b9eabc-20f4-41ef-97d8-0f565de8fd8d.xml.zip	7/29/2003 4:21 PM	ZIP	
74d2d935-b93a-4261-98bb-b7c1f5f18920.xml.zip	7/31/2003 12:45 PM	ZIP	
807ba08c-953c-4216-8ef4-676c01ed444c.xml.zip	7/31/2003 12:06 PM	ZIP	
9d56ae58-5615-40c7-9153-50347426b709.xml.zip	7/31/2003 9:11 AM	ZIP	
cdx.wsdl	7/31/2003 10:33 AM	XML	test
dc392613-2373-46fc-b422-b6ed184dbbd8.xml.zip	7/31/2003 12:44 PM	ZIP	
name2	7/31/2003 1:34 PM	BIN	FRS
name2	8/4/2003 9:57 AM	BIN	FRS
Notif.xml	7/31/2003 1:34 PM	XML	FRS
Notif.xml	8/4/2003 9:57 AM	XML	FRS
soapclient.gif	7/31/2003 10:33 AM	BIN	test

Document Details

Name:	193b7692-49a0-4c2c-9789-0f2179cefcc
Size:	369 Bytes
Data Flow:	
Data Type:	ZIP
Created On:	7/31/2003 10:33:09 AM

Transactions

User	Dataflow	Created On	Status
cdx		7/31/2003	Processed

Current status: Ready

Last modified by: MCHMARNY on 7/31/2003 5:15:18 PM

By selecting the specific document from the list at the top of the screen, the Node administrator can view and change the status of each document.

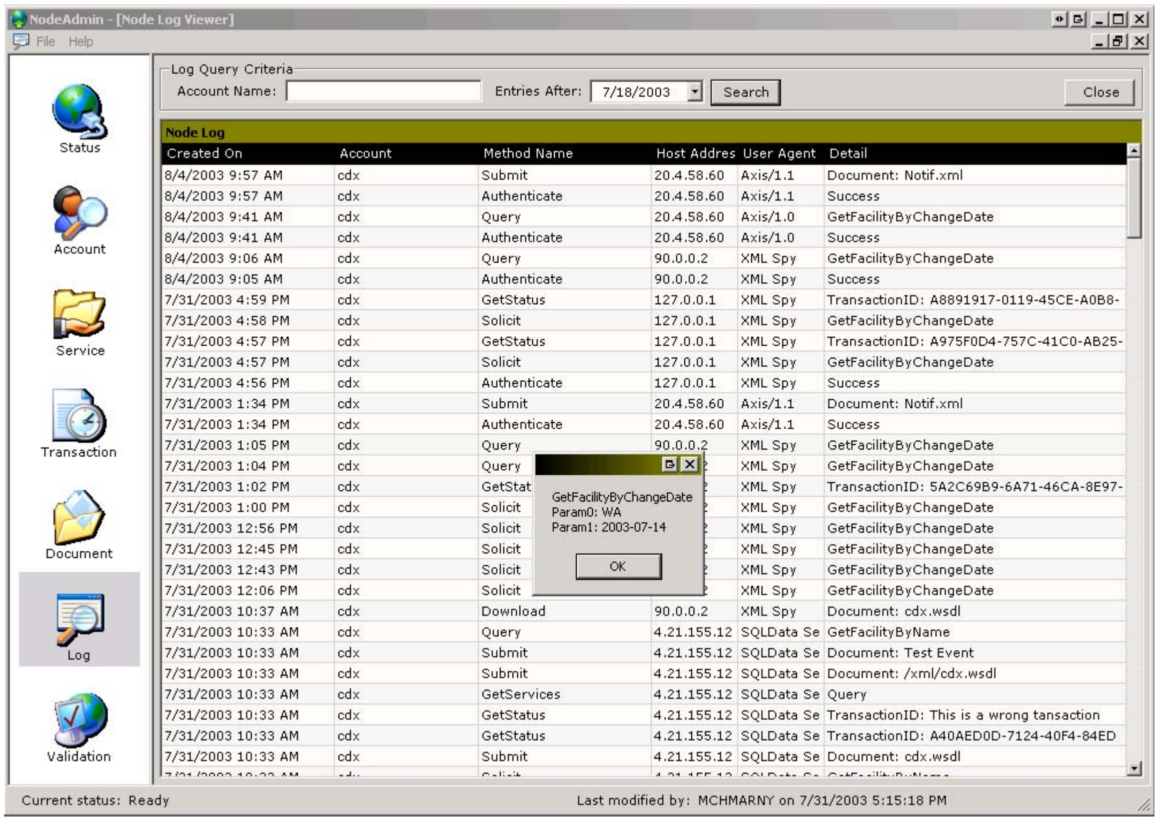
In addition, the transactions associated with this document are listed in the Transactions section in the bottom right pane.

The document detail pane provides a download button which when clicked will allow the user to download the selected document to a specific location.

In addition, using this screen, the Node administrator can associate each document with multiple transactions (requests), potentially limiting the number of documents needing to be generated for similar/identical requests.

Node Activity Log

The Node Activity Log window allows the Node Administrator to audit the node activities.



Account Name:

Entries After: 7/18/2003

Node Log

Created On	Account	Method Name	Host Addres	User Agent	Detail
8/4/2003 9:57 AM	cdx	Submit	20.4.58.60	Axis/1.1	Document: Notif.xml
8/4/2003 9:57 AM	cdx	Authenticate	20.4.58.60	Axis/1.1	Success
8/4/2003 9:41 AM	cdx	Query	20.4.58.60	Axis/1.0	GetFacilityByChangeDate
8/4/2003 9:41 AM	cdx	Authenticate	20.4.58.60	Axis/1.0	Success
8/4/2003 9:06 AM	cdx	Query	90.0.0.2	XML Spy	GetFacilityByChangeDate
8/4/2003 9:05 AM	cdx	Authenticate	90.0.0.2	XML Spy	Success
7/31/2003 4:59 PM	cdx	GetStatus	127.0.0.1	XML Spy	TransactionID: A8891917-0119-45CE-A0B8-
7/31/2003 4:58 PM	cdx	Solicit	127.0.0.1	XML Spy	GetFacilityByChangeDate
7/31/2003 4:57 PM	cdx	GetStatus	127.0.0.1	XML Spy	TransactionID: A975F0D4-757C-41C0-AB25-
7/31/2003 4:57 PM	cdx	Solicit	127.0.0.1	XML Spy	GetFacilityByChangeDate
7/31/2003 4:56 PM	cdx	Authenticate	127.0.0.1	XML Spy	Success
7/31/2003 1:34 PM	cdx	Submit	20.4.58.60	Axis/1.1	Document: Notif.xml
7/31/2003 1:34 PM	cdx	Authenticate	20.4.58.60	Axis/1.1	Success
7/31/2003 1:05 PM	cdx	Query	90.0.0.2	XML Spy	GetFacilityByChangeDate
7/31/2003 1:04 PM	cdx	Query	2	XML Spy	GetFacilityByChangeDate
7/31/2003 1:02 PM	cdx	GetStat	2	XML Spy	TransactionID: 5A2C69B9-6A71-46CA-8E97-
7/31/2003 1:00 PM	cdx	Solicit	2	XML Spy	GetFacilityByChangeDate
7/31/2003 12:56 PM	cdx	Solicit	2	XML Spy	GetFacilityByChangeDate
7/31/2003 12:45 PM	cdx	Solicit	2	XML Spy	GetFacilityByChangeDate
7/31/2003 12:43 PM	cdx	Solicit	2	XML Spy	GetFacilityByChangeDate
7/31/2003 12:06 PM	cdx	Solicit	2	XML Spy	GetFacilityByChangeDate
7/31/2003 10:37 AM	cdx	Download	90.0.0.2	XML Spy	Document: cdx.wsdl
7/31/2003 10:33 AM	cdx	Query	4.21.155.12	SQLData Se	GetFacilityByName
7/31/2003 10:33 AM	cdx	Submit	4.21.155.12	SQLData Se	Document: Test Event
7/31/2003 10:33 AM	cdx	Submit	4.21.155.12	SQLData Se	Document: /xml/cdx.wsdl
7/31/2003 10:33 AM	cdx	GetServices	4.21.155.12	SQLData Se	Query
7/31/2003 10:33 AM	cdx	GetStatus	4.21.155.12	SQLData Se	TransactionID: This is a wrong tansaction
7/31/2003 10:33 AM	cdx	GetStatus	4.21.155.12	SQLData Se	TransactionID: A40AED0D-7124-40F4-84ED
7/31/2003 10:33 AM	cdx	Submit	4.21.155.12	SQLData Se	Document: cdx.wsdl

GetFacilityByChangeDate
Param0: WA
Param1: 2003-07-14

Current status: Ready

Last modified by: MCHMARNY on 7/31/2003 5:15:18 PM

Double clicking any individual event displays its details in a pop-up box.

Note that this log represents only successful requests and their responses. All faults and errors generated by the Node application are logged in the Node application event log and can be accessed though the Windows Event Log Utility.

Node Security

Authentication

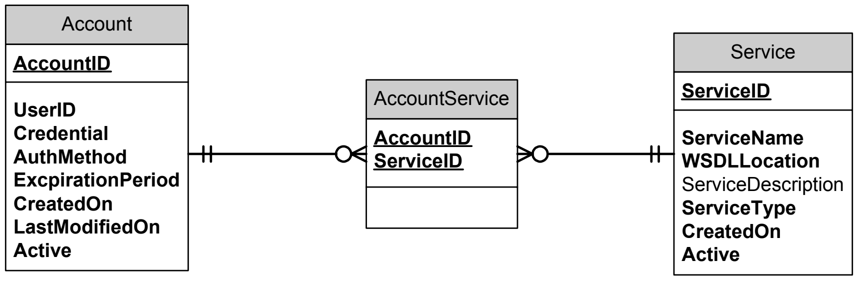
The Authenticate Web method authenticates a user using a supplied credential. It returns a security token when successful. The security token is than included in all other method invocations, except the NodePing method, as a proof of identity.

A security token is an opaque string that is meaningful only to the issuer or trusted peers. The Node application associates each issued security token to the UserID for token verification, and to a timestamp for aging and expiration checking.

The Node implements an aging strategy to prevent replay attack. An expired token is discarded immediately and any following request providing that token will be rejected. The token life span is defaulted to ten minutes, but can be modified for each user using the NodeAdmin utility.

Authorization

The Ecology Node provides basic user/service authorization at the account level managed through using the Account interface in the NodeAdmin tool. The database diagram below shows the structure of the data used to administer authorization.



Audit (Logging)

The Node logs received transactions in a persistent storage (database table) area and provides search capability using the Node Log interface in the NodeAdmin utility. This functionality allows for tracking transactions either by date or requestor user identifier. In addition to information about submitted requests and documents, the log records information pertaining to the major milestones of each user session.

In addition, the Node provides a detailed Input/Output trace log that contains detailed processing steps for each user.

Establishing Data Flows

This section describes the basic steps required to set up a new data flow processor in the Ecology Node. As the Network becomes the preferred data exchange mechanism, additional data flows will be made available through the Node.

Data Flow Integration

To allow maximum flexibility, the Node application allows for loosely-coupled assemblies to perform the actual specific data flow request processing. To make this possible the Node architecture defines some basic rules for development of the data flow processing assemblies:

1. There should be only one flow processor per flow. The main assembly (DLL) implementing the IDataFlowRequestProcessor interface may reference other assemblies but it must do so independently of the main Node processes.
2. Each flow processing assembly has to implement the IDataFlowRequestProcessor interface. (A compiled version of the interface has been provided along with this document)

In addition, the Node Web.config files must reference the individual flow request processing assemblies using the following standard:

Key:	The name of the data flow. This must match the name associated with the service.
Value:	Comma delimited string containing the following information:

i. Name of the DLL for this request processor without the extension

ii. Fully qualified name of the class implementing the interface

Data Flow Development

1) Develop DTS package(s) to replicate data into the Node database

The data should be transformed from its originating database structure and format to match that of the new data flow schema. This data replication should occur on a regular frequency via a SQL Server batch job. This approach simplifies the XML formulation when serving data, and improves the performance of the Web services. Tables created in the Node Database should be named with a consistent prefix (e.g., FRS) to differentiate them from those used to support other data flows.

2) Develop Transact SQL (TSQL)

a) Create a stored procedure

Create a single stored procedure that selects all necessary data to fulfill the dataflow schema.

```
SELECT * FROM table1 /* parent table */
SELECT * FROM table2 /* first child */
SELECT * FROM table3 /* second child */
```

b) Apply record filtering using criteria parameters

Use parameters to filter the results. For performance reasons it may be beneficial to perform parameter filters on a local variable of type table and then use the IN keyword to filter the sub selects.

```
DECLARE @t TABLE (rowid int identity, pkId varchar(36))

INSERT INTO @t  (pkId)
SELECT     PrimaryKeyField
FROM       SomeTable
WHERE      SomeField = @Variable1 AND OtherField = @ Variable2
ORDER BY   PrimaryKeyField

SELECT * FROM table1 WHERE PrimaryKeyField IN (
SELECT pkId FROM @t )
SELECT * FROM table2 WHERE PrimaryKeyField IN (
SELECT pkId FROM @t )
SELECT * FROM table3 WHERE PrimaryKeyField IN (
SELECT pkId FROM @t )
```

Definition of the appropriate filtering criteria is outside of the scope of this document. The reader is referred to the SQL Server Transact SQL documentation.

NOTE: Make sure that result set for the above stored procedure has a single field, common across all tables within the dataset that corresponds to the primary key.

c) Test the Query

Use the SQL Query Analyzer to test the stored procedure to make sure that the correct data is returned.

3) Generate data class

a) Use xsd.exe utility to generate a .Net class

The .Net Framework includes a utility (xsd.exe) which simplifies generation of a class from an XML Schema which later will be serialized to an XML document. Using this utility guarantees that the resulting object (when serialized) will conform to the originating schema.

The reader is directed to the following link for details on the use of xsd.exe:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpconxmlschemadefinitiontoolxsdexe.asp

b) Modify the resulting class

The class generated by the xsd.exe is a generic object which needs to be modified to fit the data flow project. The main parts which will have to be modified for the purposes of the Node are the namespace and library references. In addition to the default references the new class should contain the following references:

```
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml;
using System.Xml.Serialization;
using System.Xml.Schema;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters;
using System.Runtime.Serialization.Formatters.Soap;
namespace Windsor.Web.NodeEngine
```

4) *Expose query methods*

a) Implement IDataFlowRequestProcessor

In order for the Node Query and Solicit methods to begin using this new dataflow, the new class has to implement the IDataFlowRequestProcessor interface. The dataflow processing class should overlay the ProcessRequest method.

```
string ProcessRequest(
     string connectionString,
     string serializationPath,
     string methodName,
     int rowId,
     int maxRows,
string[] parameters);
```

It is the responsibility of the dataflow processing object to populate the resulting class with data using the provided connection string and serialize that object to an XML file in the appropriate location.

The use of the rowed and maxRows allows the requestor to page records. See the *Network Node Functional Specification Version 1.1* for specific uses of these parameters.

b) Populate relationship

The PopulateRelationships method should be invoked, passing the dataset developed in the first step by reference together with the name of the primary key.

```
PopulateRelationship(ref ds, pkName);
```

5) *Populate object*

a) Create an instance of the return object

Create an instance of the object to be returned. The instance then will be populated with the dataset data.

```
NameOfTheNewClass obj = new NameOfTheNewClass();
```

b) Populate primitive properties

Load the primitive properties of the object with appropriate data.

```
Obj.Property1 = Tables[n][i];
```

c) Enumerate Tables/Rows

Load the multiple instance properties data using the Table/Row enumeration methods

```
foreach (DataRow dr in Tables[n].Rows) {
     SubObject subObj = new SubObject ();
     subObj.Property1 = DataSet.Tables[n][i];
Obj.ArrayProperty.Add(subObj);
}
```

6) Assign a new service to the Node

Having developed the data class which will be responsible for all data queries in this dataflow, the next step is to add this class to the Node project to allow for Query and Solicit methods to be able to execute it.

The Query and Transaction classes provide detailed instructions on integrating new data flows in to the Node.

7) Publish the updated Node

When all the above steps are complete, publish the new service using the *NodeAdmin* tool by specifying the new interface's WSDL file location.

8) Testing

To validate the successful deployment of the Node application, it is recommended that two sets of tests are performed. An Internal Test to validate the initial deployment and an external test, using the on-line tool provided by CDX.

Internal Tests

In order to allow for quick self-test using the auto-generated browser interface, the Node *Web.config* file can be modified as follows:

```
<configuration>
  <system.web>
    <webServices>
      <!-- Remove this comment prior to going to production
      <protocols>
        <clear/>
        <add name="HttpSoap"/>
        <add name="Documentation"/>
      </protocols>
    -->
    </webServices>
  </system.web>

</configuration>
```

Once the *Web.config* file is modified, a browser interface can be used to test each individual interface.

An alternative approach to testing the Node deployment is to use the NodeWinClient utility developed by Windsor and freely available for download at http://www.windsorsolutions.biz/nodeclient/.

External Tests

The EPA CDX on-line test application provides the ability to test any Node in the Network, by triggering Network WSDL-compliant requests on that Node. If a Node passes a test with this tool, if is very likely, if not guaranteed, that the Node will be interoperable with other Network WSDL-compliant Nodes. This tool, which is intended to verify general compliance with the *Network Node Functional Specification Version 1.1*, focuses on interoperability among Nodes.

The EPA Node Test Site can be accessed at https //test.epacdxnode.net/test/.

Glossary

Term	Definition
.NET Framework	The .NET Framework is a component of the Windows operating system that provides the programming model for building, deploying and running Web-based applications, smart client applications and Web services. The .NET Framework consists of the common language runtime (CLR) and a unified class library.
ADO.NET	The suite of data access technologies included in the .NET Framework class libraries
ASP.NET	The development component for building server-based Web applications. An evolution of ASP into the .NET Framework.
assembly	The primary building block—also the unit of deployment and versioning—of a .NET Framework application. An assembly includes an assembly manifest, which describes the contents of the assembly
C#	A new ECMA-approved programming language designed for the .NET Framework. C#, which is an evolution of C and C++, is type safe and object oriented. Because it is compiled as managed code, it benefits from the services of the common language runtime, such as language interoperability, enhanced security, and garbage collection.
class library, .NET Framework	A library of classes, interfaces, and value types that are included in the Microsoft .NET Framework and can be used from any CLS-compliant language. The .NET Framework class library provides access to system functionality and is designed to be the foundation on which .NET Framework applications, components, and controls are built.
common language runtime (CLR)	The engine at the core of .NET Framework-managed code execution. The runtime supplies managed code with services such as cross-language integration, code access security, object lifetime management, and debugging and profiling support.
Extensible Markup Language (XML)	A subset of Standard Generalized Markup Language (SGML) that is optimized for delivery over the Web. XML provides a uniform method for describing and exchanging structured data that is independent of applications or vendors.
garbage collection (GC)	The process of transitively tracing through all pointers to actively used objects to locate all objects that can be referenced and then arranging to reuse any heap memory that was not found during this trace. The CLR garbage collector also compacts the memory that is in use to reduce the working space needed for the heap.
HTTP	Hyper Text Transfer Protocol is a standard Internet protocol for transfer of information between servers and between clients and servers.
loosely coupled architecture	A distributed application in which you can change the implementation of one tier without affecting any of the other tiers. Contrast tightly coupled

Term	Definition
	architecture.
managed code	Managed code supplies the metadata necessary for the CLR to provide services, such as memory management, cross-language integration, code access security, and automatic lifetime control of objects. All code based on MSIL executes as managed code.
metadata	Data (or information) about data. Many different systems use metadata—for example, type libraries in COM provide metadata and databases have schemas. In the CLR, metadata is used to describe assemblies and types. It is stored with them in the executable files, and is used by compilers, tools, and the runtime to provide a wide range of services. Metadata is essential for runtime type information and dynamic method invocation.
native code	Code that has been compiled to processor-specific machine code.
n-tier	System architecture that separates presentation, business logic, data access, and database (or other persistence mechanism) tiers.
reflection	.NET Framework technology that allows you to examine metadata that describes types and their members. Reflection can be used to create, invoke, and access type instances at run time.
serviced component	The mechanism that enables COM+ services to be available to .NET Framework classes.
side-by-side execution	The ability to run multiple versions of the same assembly simultaneously. This can be on the same computer or in the same process or application domain. Allowing assemblies to run side-by-side is essential to support robust versioning in the common language runtime. Side-by-side is also used to describe to describe two versions of the .NET Framework running simultaneously on the same computer.
SOAP	Simple Object Access Protocol, a W3C standard. A lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML-based protocol for exchanging structured and type information on the Web. The SOAP protocol contains no application or transport semantics, which makes it highly modular and extensible.
tightly coupled architecture	A distributed application where a change to any tier affects some or all the other remaining tiers. Contrast loosely coupled architecture.
UDDI	Universal Description, Discovery, and Integration (UDDI) specification. An initiative that creates a global, platform-independent, open framework to enable Web service providers to advertise the existence of their Web services and for Web service consumers to locate Web services of interest.
Web services	A programming model that provides the ability to exchange messages in a scalable, loosely coupled, and platform-neutral environment using standard protocols such as HTTP, XML, XSD, SOAP, and WSDL. The SOAP-based XML messages exchanged between a Web service and its clients can be structured and typed, or loosely defined. The flexibility of using a text format such as XML enables the message exchange to evolve over time in a loosely

Term	**Definition**
	coupled way. Because they are based on standard protocols and are platform neutral, Web services enable communication with a broad variety of implementations, platforms, and devices.
Web Services Description Language (WSDL)	An XML-based contract language for describing network services offered by a server.
Windows Forms	A rich Windows client library that encapsulates native Win32 APIs and exposes secure, managed classes for creating smart Windows client applications. The Windows Forms class library provides many controls, such as buttons, check boxes, drop-down lists, combo boxes, data grid, and others, that encapsulate user-interface and other client-side functionality.
WSDL	(see Web Services Description Language)
XML	(see Extensible Markup Language).
XML Schema Definition (XSD)	A W3C Recommendation that specifies how to formally describe the elements of an XML document. The schema can be used to verify the conformance of elements in an XML document.