

# EN-NODE DATA FLOW DEVELOPER'S GUIDE

---

**Version: 2.4**

**October 2, 2012**



1368 How Lane  
North Brunswick, New Jersey 08902  
[www.enfotech.com](http://www.enfotech.com)

## Restriction on Use and Disclosure of Document Information

This document includes data that should not be disclosed outside the business entity for which it was intended, indicated as the recipient on this title page. The entire document is copyrighted by enfoTech and is protected under the US copyright law and international treaties. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without express written permission from enfoTech & Consulting Inc.

Copyright © 2001 – 2012 by enfoTech & Consulting Inc. All Rights Reserved.

**Revision History**

<b>Version</b>	<b>Date</b>	<b>Created By</b>	<b>Reviewed By</b>	<b>Description</b>
2.4	10/02/2012	Charlie Tsai	Charlie Tsai	Updated to company contact information

## Tables of Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>4</b>
1.1	DOCUMENT PURPOSE.....	4
1.2	TERMS AND DEFINITIONS .....	4
1.3	OVERVIEW OF EN-NODE COMPONENTS .....	5
1.4	AVAILABLE WEB SERVICES .....	6
1.5	OVERVIEW OF HOW EN-NODE PROCESSES WEB SERVICE REQUESTS .....	7
<b>2</b>	<b>ADDING A DATA FLOW TO EN-NODE .....</b>	<b>9</b>
2.1	CREATE DOMAIN.....	11
2.2	CREATE OPERATION .....	14
2.3	CREATING FLOWS FROM THE DATA FLOW WIZARD .....	22
2.3.1	<i>External API Data Flow Component</i> .....	24
2.3.2	<i>Composer Data Flow Component</i> .....	27
2.3.3	<i>Populator Data Flow Component</i> .....	29
2.3.4	<i>Validator Data Flow Component</i> .....	30
2.3.5	<i>XSLT Transformer Data Flow Component</i> .....	31
2.3.6	<i>DB Procedure Data Flow Component</i> .....	32
2.3.7	<i>Email Data Flow Component</i> .....	33
2.3.8	<i>Probe Data Flow Component</i> .....	34
2.3.9	<i>Encrypt Data Flow Component</i> .....	35
2.3.10	<i>Decrypt Data Flow Component</i> .....	36
2.3.11	<i>Zip Data Flow Component</i> .....	37
2.3.12	<i>Unzip Data Flow Component</i> .....	38
2.3.13	<i>Web Services Data Flow Component</i> .....	39
2.3.14	<i>Save to Node Data Flow Component</i> .....	41
2.4	WRITING CUSTOM WEB SERVICE OR TASK SERVICE CLASS.....	41
2.4.1	<i>.Net Version:</i> .....	41
2.4.2	<i>Java Version:</i> .....	45
<b>3</b>	<b>APPLICATION PROGRAM INTERFACES (APIS).....</b>	<b>50</b>
3.1	<i>.NET VERSION:</i> .....	50
3.2	<i>JAVA VERSION:</i> .....	51
<b>4</b>	<b>NODE DATABASE DETAILS.....</b>	<b>54</b>
4.1	CONFIGURING DATABASE CONNECTION SETTINGS .....	54
4.1.1	<i>.Net Version:</i> .....	54
4.1.2	<i>Java Version:</i> .....	54
<b>5</b>	<b>MANUALLY UPDATE NODE WIZARD CONFIGURATION FILE (JAVA VERSION) .....</b>	<b>54</b>
5.1	DOWNLOAD WIZARD BEPL CONFIGURATION FILE .....	54
<b>6</b>	<b>DATA STRUCTURE.....</b>	<b>57</b>
6.1	DATA DICTIONARY.....	57
6.2	DATABASE DIAGRAM .....	63

---

## 1 Introduction

### 1.1 Document Purpose

The purpose of this document is to provide instructions on how to develop data flows and plug them into the EN-Node software. A companion document, called the **EN-Node Administrator's Guide**, is also available and provides instructions on general node and data flow administration. Node Administrators looking for instructions on how to configure and administer the node should use the **Data Flow Developer's Guide**.

### 1.2 Terms and Definitions

In this document the following terms and acronyms are used:

General Terms	
Term	Definition
Web Method	The processing that a Web Service performs when invoked. In this document, it refers to one of the ten Web Methods that all Exchange Network Nodes implement as defined by the Node Specifications and Protocol Versions 1.1 and 2.0.
Handler	A set of code that executes the logic for a particular Web Method. There is one handler for each Web Method.
Operation	A specific implementation of a Web Method, usually defined by the set of parameters passed in to a particular Web Method that differentiates the type of request. <i>(For example, the Submit Web Method has the signature (&lt;securityToken&gt;, &lt;transactionID&gt;, &lt;dataflow&gt;, &lt;flowOperation&gt;, &lt;recipient&gt;, &lt;notificationURI&gt;, &lt;documents&gt;). Each allowable setting for the dataflow parameter (e.g. - NEI, NEICritHap, FRSMonthly, etc) defines a unique <b>operation</b> for the Submit <b>Web Method</b>.)</i> As such, each Handler can have many Operations.
Domain	A logical grouping of related Operations on which to base data flow administration through the EN-Node Administration console (e.g.: NEI, FRS, RCRA, etc). Every operation is tied to one and only one Domain.
Task	A service that has an associated schedule which determine the interval at which it is executed. Tasks are executable and can be executed either on a recurring schedule using the Node Admin Console, or on demand by a Node or Domain Administrator.
NAAS	Network Authentication and Authorization Services. A set of centralized security services hosted by EPA for performing authentication and service authorization of Node users.

### 1.3 Overview of EN-Node Components

EN-Node is a software suite that provides a one-stop solution for creating, deploying, managing, scheduling, and interacting with Web service-based Exchange Network data flows. As a Web services engine, it allows other trading partners to interact with your environmental data in a secure and consistent manner. The diagram below shows the EN-Node software suite components:

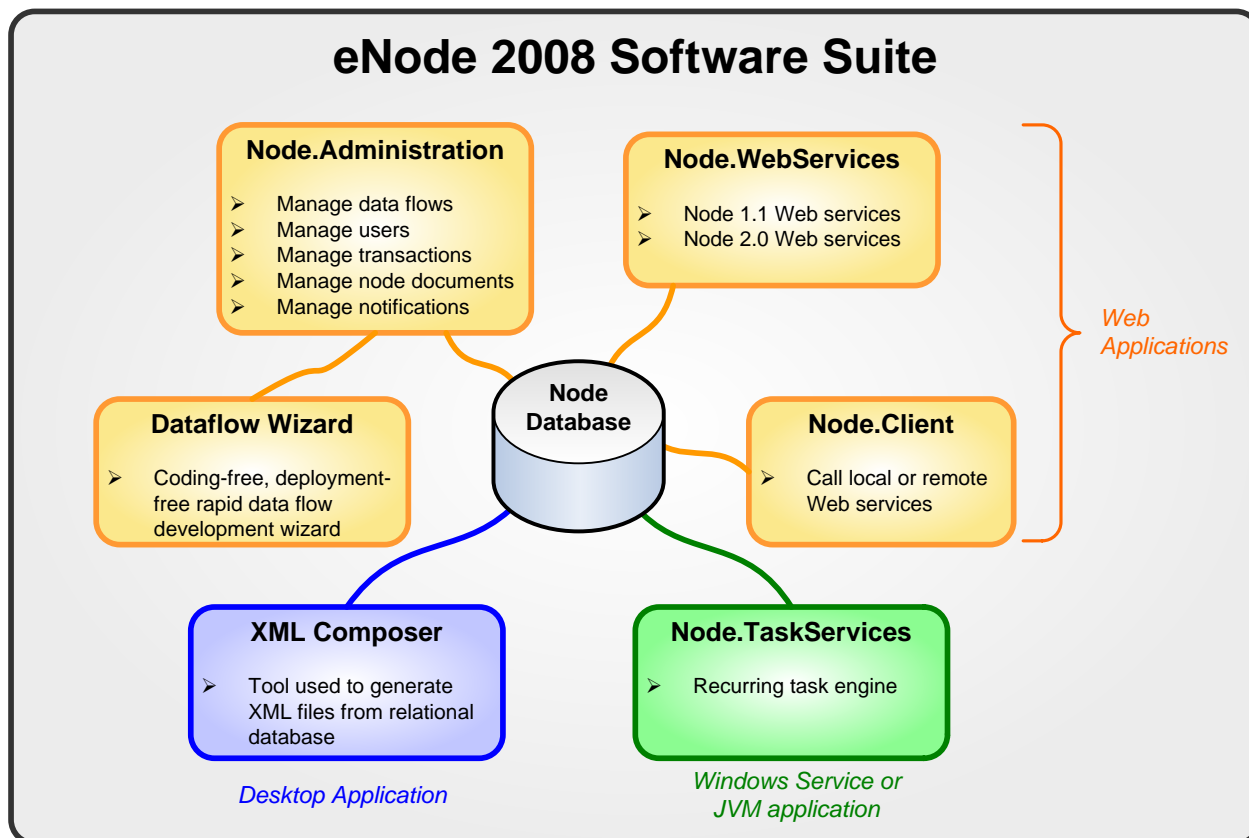


Figure 1-1 Components of Node

Each of these components is described briefly below:

- **Node.Administration:** A Web interface that allows Node and Domain administrators to configure the Node and manage data flows. The Node.Administration application serves as an interface to configure the Node.WebServices, Node.TaskServices, Node.Client, and (if applicable) Dataflow Wizard applications. This component is described in further detail in the EN-Node Administrator's Guide.
- **Node.WebServices:** A Web Service engine that controls the logic for responding to Web Service requests on the Node, providing the web services outlined in the Node 1.1 and 2.0 Specifications. When responding to a Web service request, Node.WebServices will execute logic plugged in for a particular data flow.
- **Node.Client:** A simple Web interface that allows individuals to invoke Node 1.1 or 2.0 Web Services on any Node, including the EN-Node. This application can be useful for either testing your Node functionality, or can serve as a simple Node client to invoke Web services on other Nodes. This component is described in further detail in the EN-Node User's Guide.

- **Node.TaskServices:** Provides the capability to execute tasks on a scheduled basis, which allows you to schedule and initiate Web service exchanges. These scheduled tasks typically involve the invocation of Web Services on other Nodes, such as EPA's Node. The scheduled tasks are defined by the task plug-in and are configured by a Domain Admin for a particular data flow.
- **Dataflow Wizard:** A new optional add-on for EN-Node that allows data flow developers to rapidly create new dataflows with minimal coding and deployment required by using a drag-and-drop interface to select the actions to be performed in the data flow.
- **XML Composer:** A new optional add-on for EN-Node that provides a user interface to map XML data structures such as XML schema or XML instance files to database structures to allow rapid development of XML composition or decomposition procedures.

## 1.4 Available Web Services

The Node comes preconfigured with ten standard Web services as defined in the Node 1.1 and 2.0 specifications. These Web services include<sup>1</sup>:

- **Authenticate:** Authenticates a user using a supplied credential.
- **Download:** Provides a means for retrieving documents from a Node. These can either be documents associated with a previous transaction or general documents available for download.
- **GetServices:** The GetServices method is a function in the Admin interface. It allows requesters to query services provided by a Network Node.
- **GetStatus:** Provides transaction tracking of web service requests. Once submitted, a transaction enters into different processing stages. The GetStatus method offers a web service requester a way of querying the current state of the transaction.
- **NodePing:** A utility method for determining whether a Node is accessible.
- **Notify:** The Notify method has three (3) intended uses: document notification, event notification, and status notification
- **Query:** This method is intended to run a series of predefined information requests that return data in an XML instance document that conforms to a predefined standard schema.
- **Solicit:** The Solicit method performs a requested operation asynchronously in the background. It is designed especially for queries that may take a long time.
- **Submit:** Provides a generic way of sending one or more payloads to a service provider.
- **Execute:** The Execute method is a generic web service extension that allows a node to offer additional services

---

<sup>1</sup> For more information on the Node Web Services, please refer to the Node Version 1.1 or 2.0 Specifications at: <http://www.exchangenetwork.net>.

---

## 1.5 Overview of How EN-Node Processes Web Service Requests

When a web service request is received by the Node, it is processed as indicated in the diagram below:

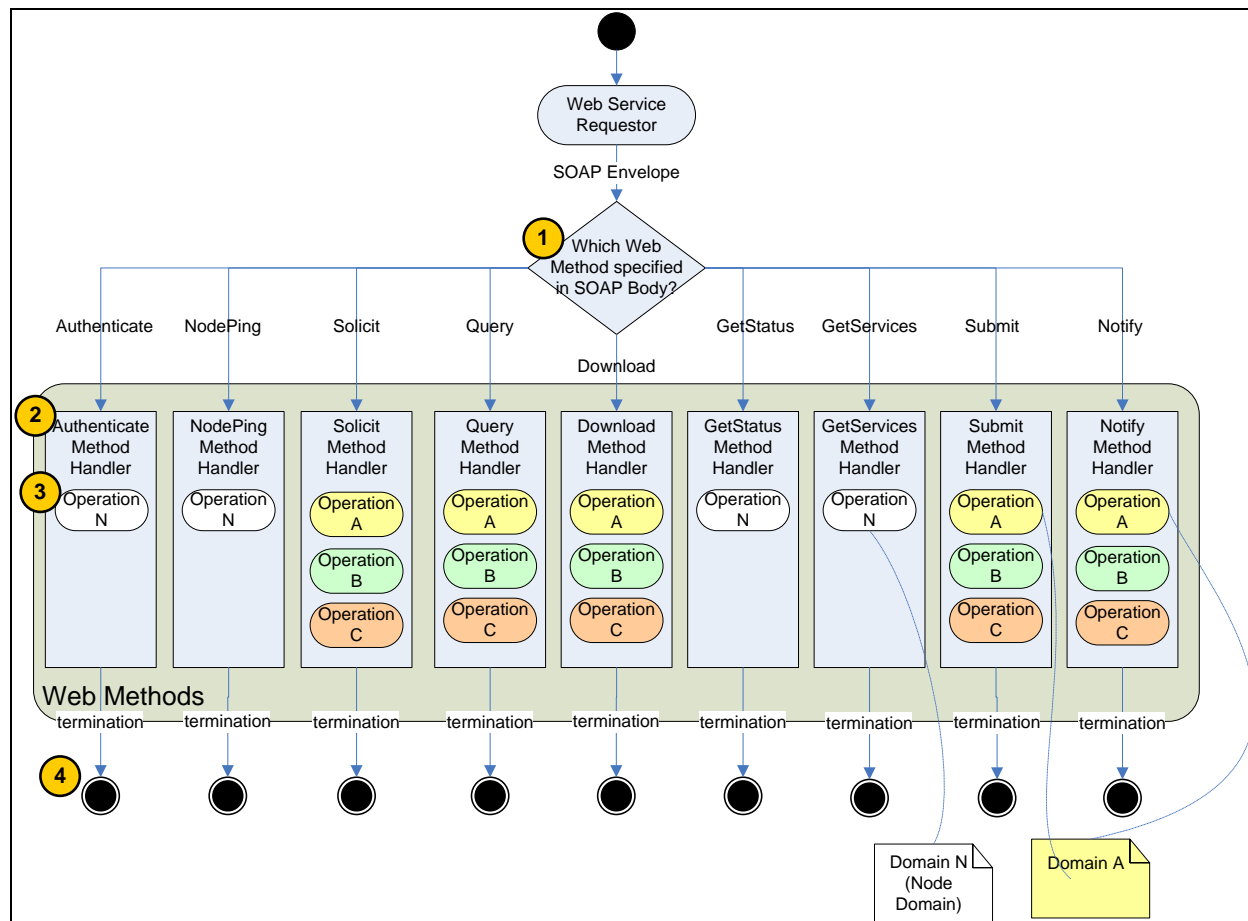


Figure 1-2: EN-Node Overview

- 1 When a request for any of the Web services listed in Section 1.4 is received at the Node address, the Node will read the Web Method identified in the SOAP body.
- 2 Based on the Web Method that is specified, the request will be routed to the appropriate Handler. The “handler” determines the logic/approach by which a Web Service request will be handled.
- 3 Within each Handler, multiple Operations can be defined that execute a specific sequence of logic. These Operations are defined by and organized by Domain Administrators.
- 4 After the operation is completed, a response is returned to the Web service requester.

Although each of the ten Web Services has its own Handler, they are all derived from one generic base Handler. The Generic Handler contains the generic logic that EN-Node executes regardless of the Web Service that is invoked. The generic handler has the following behavior:

- Generates a transaction ID each time a new Web Service request is made on the EN-Node
- Initiates logging of the Web Service request to track the status of the request
- Attempts to authorize the request to ensure that the person making the request has the appropriate security rights. The user could either be a locally managed user or NAAS-managed user. EN-Node distinguishes the two types of users by issuing different prefixes for their security token. During authorization, if the token is expired, then the authorization request will fail.
- Once authorization is completed, the EN-Node searches for and executes logic in a particular order. The order in which a dataflow is executed depends on how that dataflow was created:
  - **Dataflows created using the Dataflow Wizard:**
    - The dataflow is executed in the order as defined in the Data Flow Wizard.
  - **Dataflows created using traditional Plug-In Approach:**
    - All Plug-Ins registered as “Pre-Processes” for the Operation are executed in sequential order
    - EN-Node then executes the registered Process for the Operation and returns the invocation response value to the requestor as per Node specifications.
    - Once the Web Service method has been processed, EN-Node then executes in order any Post-Processes registered for the Operation.



## 2 Adding a Data Flow to EN-Node

To add data flows to eNode, you will need to access the Node Administration Console. It can be run by loading the following page:

- .NET: <http://<<InstallLocation>>/Node.Administration>
- Java: <http://<<InstallLocation>>/Node.Administration/Page/Entry/Login.do>

When creating a new data flow, the first decision that will need to be made is whether you will implement this using the Data Flow Wizard or will write custom code. The pros and cons of developing the data flow using the Data Flow Wizard are listed below:

- Data Flow Wizard:
  - Pros:
    - rapid data flow development
    - no J2EE or .NET coding knowledge required
    - no need for file deployment on application server
    - migration of flows from TEST to PROD is easier
    - more easily able to share solution with other eNode customers
    - easy to change any configuration parameters
  - Cons:
    - If complex logic is required, Data Flow Wizard may not be able to accommodate this
    - Less control over data flow processing
    - Less control over processing performance

Aside from an “All Wizard” versus “All Custom” approach, a blended approach is also possible, where the Data Flow Wizard is used, but the API wizard component is invoked to handle portions of the data flow that require custom logic.

When adding a data flow, you will need to decide whether you want to use the Data Flow Wizard or write custom coding for the Data Flow plug-in. The following diagram outlines the process for adding a data flow to the Node:

---

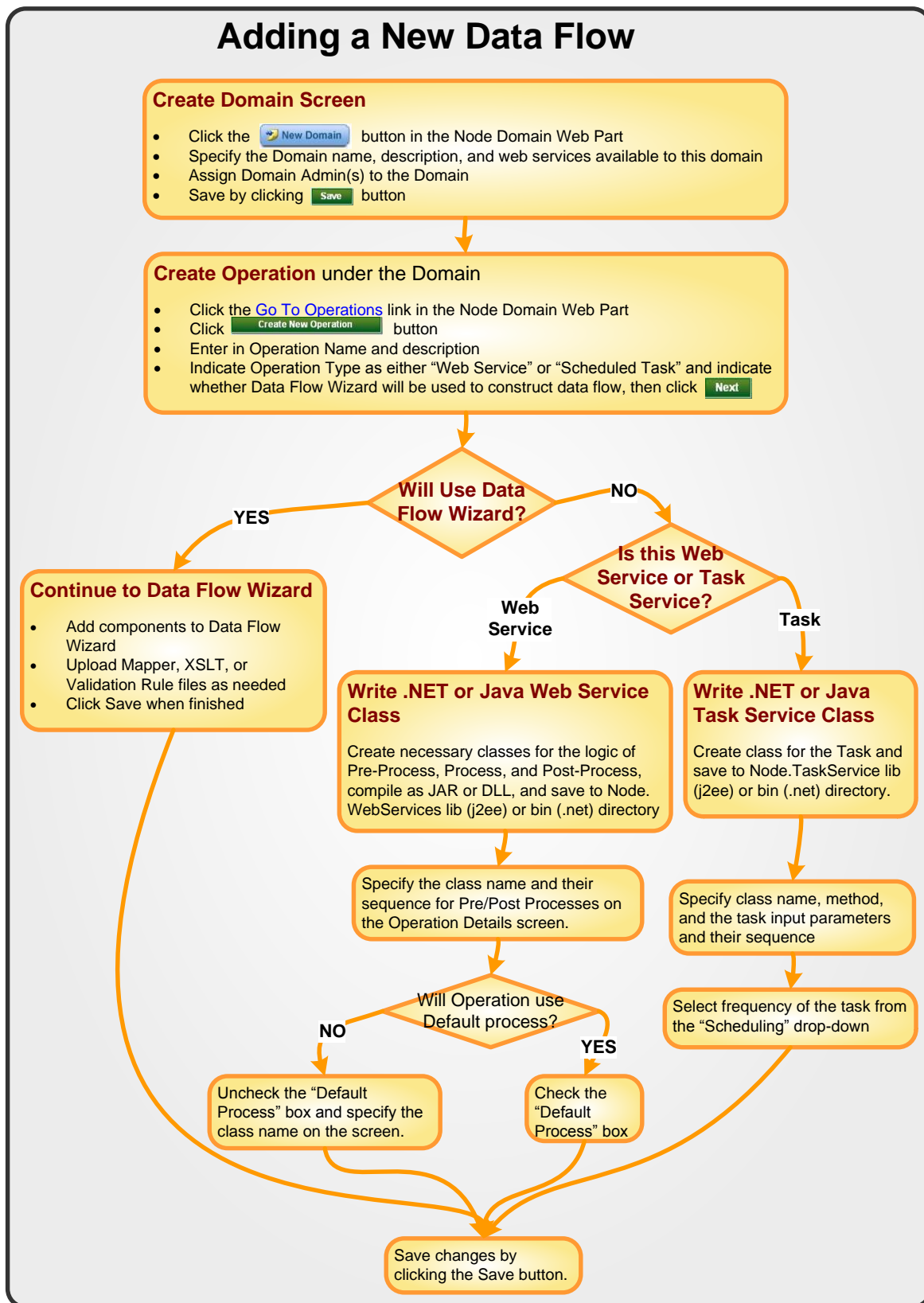


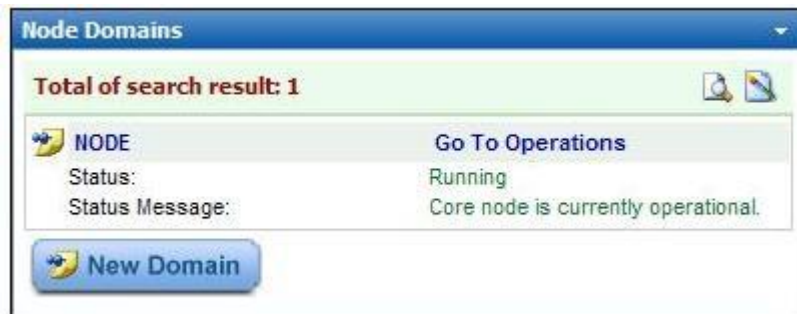
Figure 2-1 Steps for Adding a New Flow

## 2.1 Create Domain

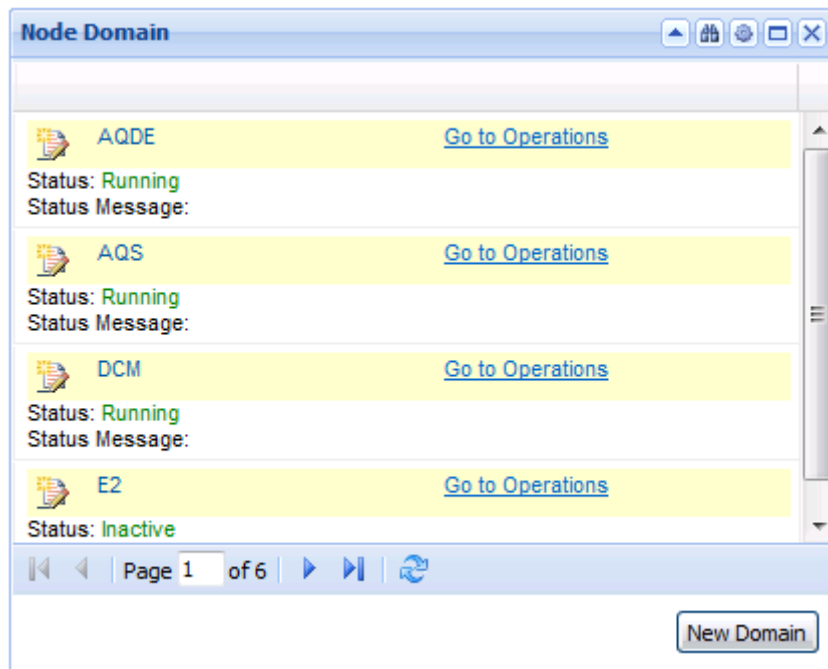
To add a new dataflow to the Node, you will first need to create a Domain. A Domain is a collection of operations (or dataflows). Operations are organized into domains to allow for operation management. *(For example, you may have 3 web service operations called `GetDrinkingWaterResults`, `GetDrinkingWaterInventory`, and `GetDrinkingWaterViolations`. By grouping these 3 operations into a domain, you can setup 1 domain administrator who would be able to manage all 3 operations.)*

To add a new domain, find the **Node Domains** Web Part in the Admin Console Dashboard. In the Node Domain Web Part, Node Administrators will be able to see all Domains, whereas Domain Administrators will only be able to see the domains they are associated with.

.Net:



Java:



**Figure 2-2 Domain Management**

Administrators can create new Domains by clicking on the



button which will bring up the following screen:

---

.Net:

New Node Domain Information

New Domain Information

Domain Name:\*

Status:

Running

Allowed Web Services:

☐ Submit

☐ Download

☐ Query

☐ Solicit

☐ Notify

Domain Status Message:

Domain Description:

Domain Administrators

Select

User Name

☐

admin

Cancel

Save

Java:

**Domain Management - New Domain**

**Domain Details** (\* Required Fields)

Domain Name: \*  Domain Status: **Running** Domain Active Status: **Active**

Allowed Web Services: ☐ Submit ☐ Download ☐ Query ☐ Solicit ☐ Notify

Domain Status Message:

Domain Description:

**Domain Admins/Privileges**


Assigned	Admin
<input type="checkbox"/>	tester1
<input type="checkbox"/>	admin
<input type="checkbox"/>	mytestuser
<input type="checkbox"/>	test2
<input type="checkbox"/>	test
<input type="checkbox"/>	charlie
<input type="checkbox"/>	devRCRAAdmin
<input type="checkbox"/>	doug
<input type="checkbox"/>	Lijie
<input type="checkbox"/>	dcm
<input type="checkbox"/>	aaa
<input type="checkbox"/>	rcra
<input type="checkbox"/>	test3
<input type="checkbox"/>	tritester
<input type="checkbox"/>	DCM

**Save**

**Figure 2-3 Create New Domain Screen**

The Domain Details page displays details about the domain:

- **Domain name:** The name of the domain assigned by the Node Administrator while creating the domain. This field is not editable once created.
  - **Domain Status:** This allows the Domain Administrator to set the Domain to either Active (Running) or Inactive (Stopped) Status.
  - **Domain Active Status (Java):** This status shows the active domains. It is useful when you want to hide inactive domains or show active domains.
  - **Allowed Web Services:** This displays the set of Web Services that a Domain Admin is allowed to create when creating operations under this domain. This allows the Node Administrator to restrict which web services a Domain Administrator can create.
  - **Domain Status Message:** This is a message that the Domain Admin can set for the Domain. This message will appear when a user views the Domain's status at the Node Status page.
  - **Domain Description:** Provides a general description of the Domain.
  - **Domain Administrators:** Allows the Node or Domain Admin to define which Admin Console users can be a Domain Admin for this domain.
-

On this screen, Administrators could fill in details for the new Domain to be created. Upon filling up all the necessary details, the Domain can be created by clicking on the  button.

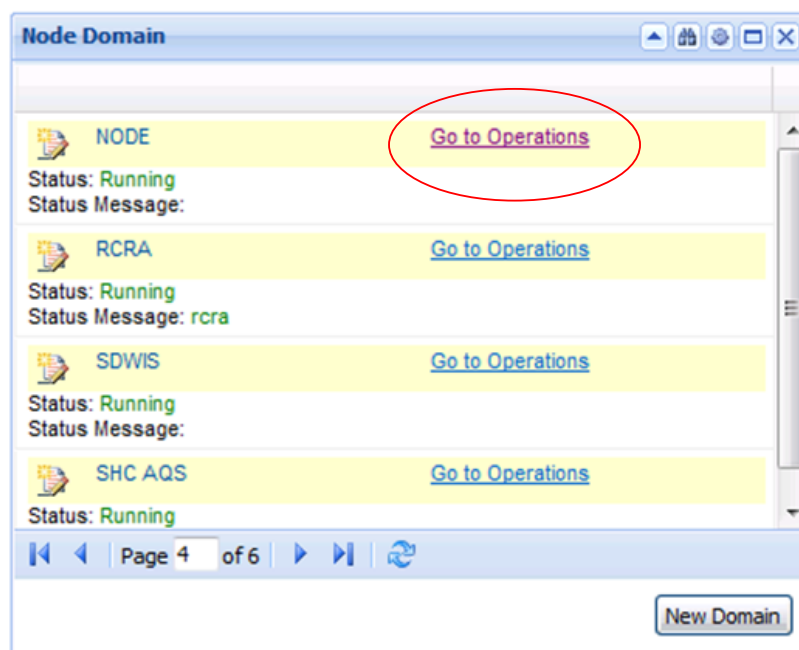
## 2.2 Create Operation

In order to add a new dataflow to the Node, you must create a new operation. To do so, click on the “Go To Operations” link in the Node Domain Web Part, as shown here:

.Net



Java:



**Figure 2-4 Domain Management - Go To Operations**

This will take the Admin to the Operations Management screen as shown below:

.Net:

**Node Operation Search Criteria**

Operation Name:

Operation Type:



Operation Status:

Web Method:

Back to DashBoard

Search

Create New Operation

View	Operation	Type	Web Service Name	Status	Status Message
	OpenDumpData2.0	WEB_SERVICE	QUERY	Running	
	testing opendump	WEB_SERVICE	QUERY	Running	testing opendump

Java:

**Operation Management - NODE**

**Searching Criteria**

Operation Name:

Operation Type:

Web Method:

Status: **Active**

Search

View/Edit	Operation Name	Operation Type	Web Service Method	Operation Status	Operation Status Message
<a href="#">Edit</a>	Cryptographic	WEB_SERVICE	AUTHENTICATE	Running	Testing Authentication/Authorization PlugIn
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	NODEPING	Running	null
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	SUBMIT	Running	null
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	NOTIFY	Running	null
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	GETSTATUS	Running	null
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	GETSERVICES	Running	null
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	EXECUTE	Running	null
<a href="#">Edit</a>	DEFAULT	WEB_SERVICE	DOWNLOAD	Running	null
<a href="#">Edit</a>	GetFacilityByName	WEB_SERVICE	SOLICIT	Running	null
<a href="#">Edit</a>	GetFacilityByName	WEB_SERVICE	QUERY	Running	null
<a href="#">Edit</a>	GetStatusTask	SCHEDULED_TASK	null	Running	GetStatusTask
<a href="#">Edit</a>	PASSWORD	WEB_SERVICE	AUTHENTICATE	Running	null


Create Operation

**Figure 2-5 Operations Management**

On this page, the Administrators can view, edit, and create Operations for the Domain. This page also allows searching for Operations through the following criteria:

- Operation Name
- Operation Type: Web Service or Scheduled Task

- Web Method used in the Operation
- Status: Running (active) or Stopped (inactive)

If any operations exist in the list, you can click on the view icon  to view/edit the operation details:

Operation Information

Operation Name:

Status:

Operation Status Message:

Operation Description:

Operation Type:

Pre Processes

Add Pre Process

Remove Selected Pre Processes

Process

Web Service:

☒ Default Process:

Post Processes

Add Post Process

Remove Selected Post Processes

Back

Delete Operation

Save

.Net:



Java:

The screenshot displays a web-based form titled "Operation Details" with a blue header bar. The form is divided into several sections:

- \* Required Fields**: A section at the top indicating required fields.
- Operation Details**: Contains fields for "Operation Name" (set to "AQDEMonitorData"), "Operation Status" (set to "Running"), "Operation Status Message" (a large text area), and "Operation Description" (a large text area).
- Operation Type**: A dropdown menu set to "WEB\_SERVICE".
- Web Service Details**: Contains a "Web Service" dropdown menu set to "QUERY".
- Pre-Process(es)**: A table with columns "Select", "Sequence", and "Class Name". Below the table are buttons "Add Pre-Process" and "Remove Selected Pre-Processes".
- Process**: Contains a "Class Name" field set to "com.enfotech.aqde.hls.nj.NodePlugIn". Below it is a table with columns "Select", "Sequence", and "Parameter Name". The table has four rows with sequence numbers 1, 2, 3, and 4. Below the table are buttons "Add Parameter" and "Remove Selected Parameters".
- Post-Process(es)**: A table with columns "Select", "Sequence", and "Class Name". Below the table are buttons "Add Post-Process" and "Remove Selected Post-Processes".
- Navigation**: At the bottom, there are three buttons: "Previous" (blue), "Save" (blue), and "Delete" (red).

Figure 2-6 Operation Details




This screen allows you to edit the following information:

- **Operation Name**: The name of the operation. This field is not editable once created.
- **Operation Status**: Set the status of the Operation as either running or stopped.
- **Operation Status Message**: Specify Operation Status Message, which will be displayed in the Node Status page.
- **Operation Description**: Specify a description of the operation

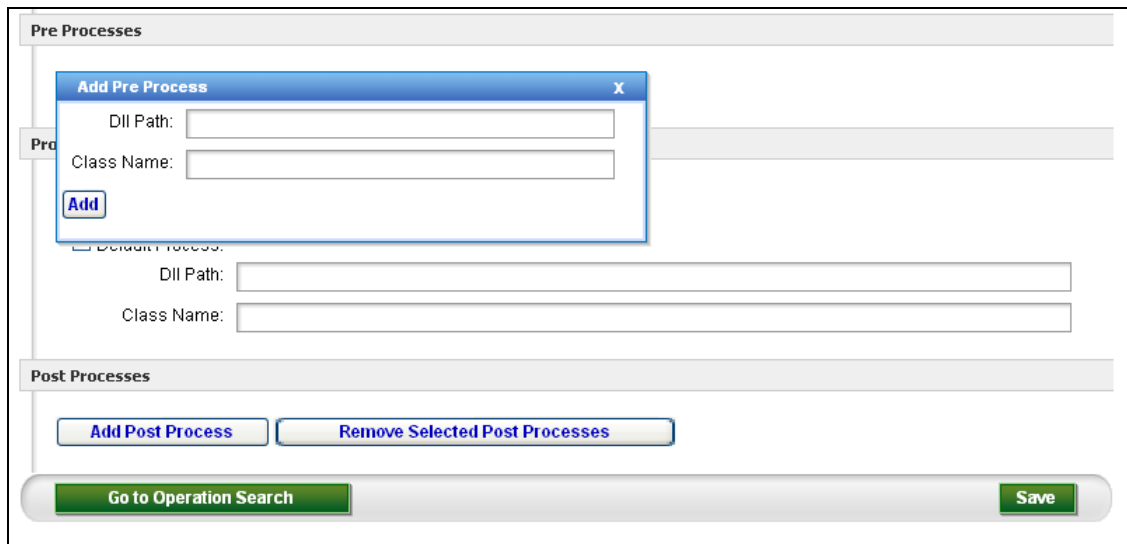
- Operation Type: Indicates whether this operation is a Web Service or Task Service.
- Web Service/Task Details:
  - Non-Data Flow Wizard Flows: Add/remove pre-processes and post-processes (for a Web Service Operation) or parameters (for a Scheduled Task Operation) if this data flow wasn't created using the Data Flow Wizard.
  - Data Flow Wizard Flows: Shows a button to load the Data Flow Wizard. [See Section 2.3 for more details.](#)

The changes made to the Operation could be saved by clicking on the  button. The Operation can be deleted by clicking on the  button.

### Non Data Flow Wizard Flows:

For a Web Service Operation, if the “Default Process” checkbox is checked, the Operation implements the “DEFAULT” Operation on the “NODE” domain. If this is unchecked, a text field will appear on the screen where the user can enter the Class name that will define the process. New pre-processes and post-processes could be added to the Operation by selecting the  or the  buttons, then entering the DLL path (for .NET Node), the class name and clicking on the  button as shown in the following diagram.

.Net:



The screenshot shows a software interface with a 'Pre Processes' section. An 'Add Pre Process' dialog box is open, featuring input fields for 'Dll Path' and 'Class Name', and an 'Add' button. Below the dialog, there are more input fields for 'Dll Path' and 'Class Name'. The 'Post Processes' section below contains 'Add Post Process' and 'Remove Selected Post Processes' buttons. At the bottom of the interface are 'Go to Operation Search' and 'Save' buttons.

Java:

Web Service Details

Web Service: \*

QUERY

Pre-Process(es)

Select	Sequence	Class Name
<input type="checkbox"/>	1	<div>com.enfotech.aqde.NodePlugIn</div>
<input type="checkbox"/>		

Add Pre-Process

Remove Selected Pre-Processes

Process

Class Name: \*

com.enfotech.aqde.hls.nj.NodePlugIn

Parameter Names

Select	Sequence	Parameter Name
<input type="checkbox"/>	1	1
<input type="checkbox"/>	2	2
<input type="checkbox"/>	3	3
<input type="checkbox"/>	4	3
<input type="checkbox"/>		

Add Parameter

Remove Selected Parameters

Post-Process(es)

Select	Sequence	Class Name
<input type="checkbox"/>		

Add Post-Process

Remove Selected Post-Processes

Previous

Save

Delete

Figure 2-7 Add Pre Process

Pre-processes and post-processes could also be removed from the Operation by selecting either the 

Remove Selected Pre Processes

 or 

Remove Selected Post Processes

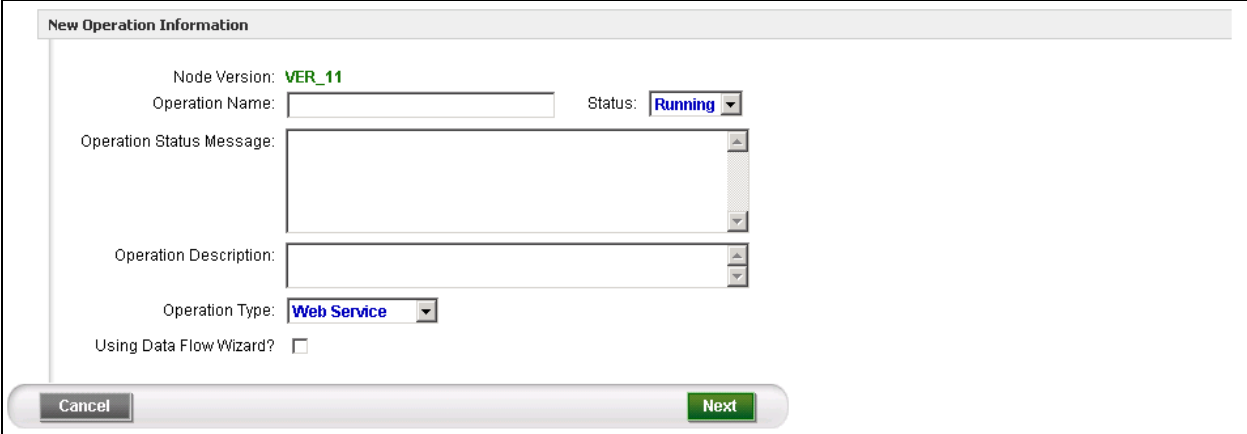
 buttons.

Administrators can create new Operations by clicking on the 

Create New Operation

 button on the Operations Management page. This will display the following screen:

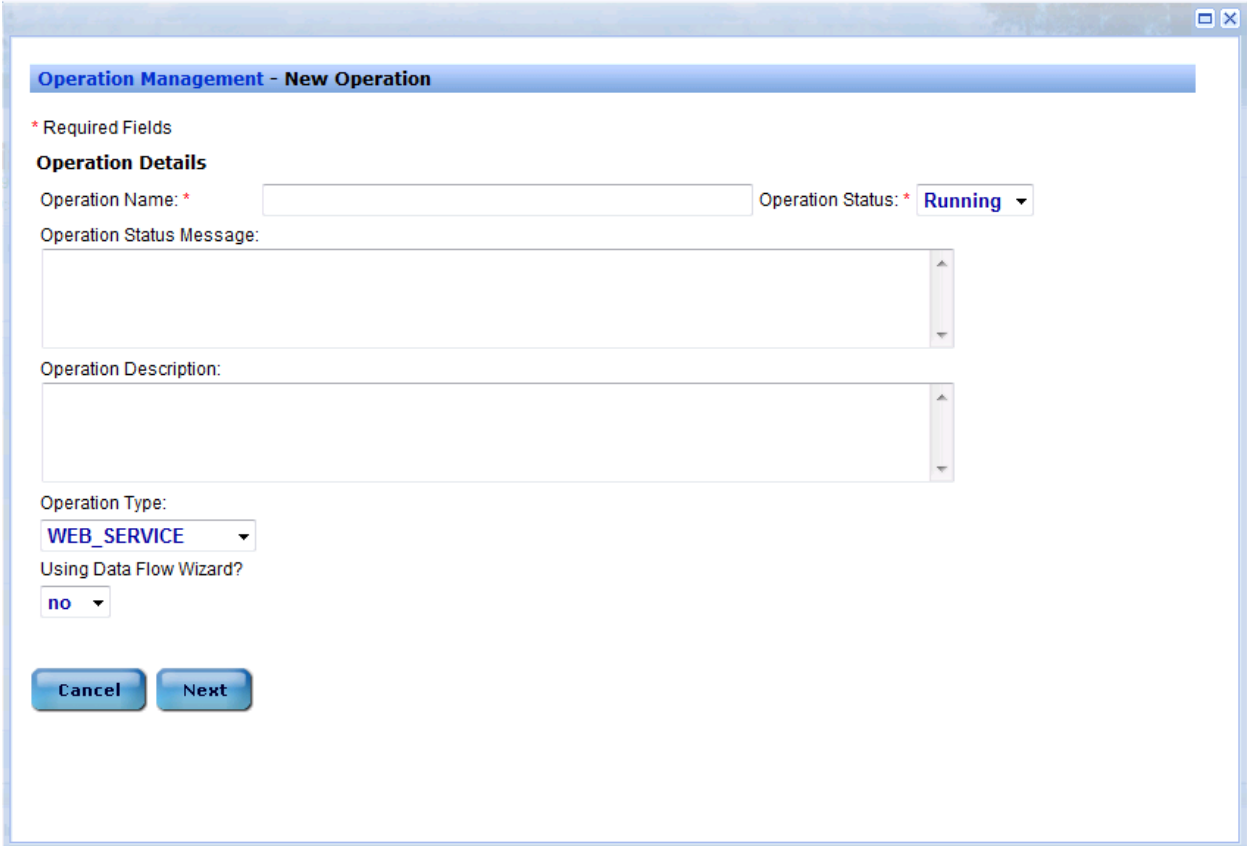
.Net:



The dialog box titled "New Operation Information" contains the following fields and controls:

- Node Version: VER\_11
- Operation Name: [Text Field]
- Status: Running (dropdown menu)
- Operation Status Message: [Text Area]
- Operation Description: [Text Area]
- Operation Type: Web Service (dropdown menu)
- Using Data Flow Wizard? ☐
- Buttons: Cancel, Next

Java:



The dialog box titled "Operation Management - New Operation" contains the following fields and controls:

- \* Required Fields
- Operation Details**
- Operation Name: \* [Text Field]
- Operation Status: \* Running (dropdown menu)
- Operation Status Message: [Text Area]
- Operation Description: [Text Area]
- Operation Type: WEB\_SERVICE (dropdown menu)
- Using Data Flow Wizard? no (dropdown menu)
- Buttons: Cancel, Next

**Figure 2-8 Create New Operation**

The New Operation Information screen allows you to enter the following information:

- **Operation Name:** The name of the operation. This field is not editable once created.
  - **Operation Status:** Set the status of the Operation as either running or stopped.
  - **Operation Status Message:** Specify Operation Status Message, which will be displayed in the Node Status page.
-

- **Operation Description:** Specify a description of the operation
- **Operation Type:** Indicates whether this operation is a Web Service or Task Service.
- **Using Data Flow Wizard:** Indicates whether or not the Data Flow Wizard is desired to create data flows.  
[See Section 2.3 for more details.](#)

Clicking on the **Next** button will show a screen to identify the process (Web Service) and:

- **For Non Data Flow Wizard Flows:** Add new and remove existing pre and post processes

**Operation Management - New Operation**

\* Required Fields

**Web Service Details**

Web Service: \* **QUERY**

**Pre-Process(es)**

Select	Sequence	Class Name
<input type="checkbox"/>		

**Add Pre-Process** **Remove Selected Pre-Processes**

**Process**

Class Name: \*

Parameter Names

Select	Sequence	Parameter Name
<input type="checkbox"/>		

**Add Parameter** **Remove Selected Parameters**

**Post-Process(es)**

Select	Sequence	Class Name
<input type="checkbox"/>		

**Add Post-Process** **Remove Selected Post-Processes**

**Previous** **Save**

- **For Data Flow Wizard Flows:** A button, **Go to Data Flow Wizard**, to the Data Flow Wizard

Operation Management - New Operation

\* Required Fields

**Web Service Details**


Web Service: \* QUERY

Parameter Names

Select	Sequence	Parameter Name
<input type="checkbox"/>		

Add Parameter Remove Selected Parameters

Previous Goto Wizard Save

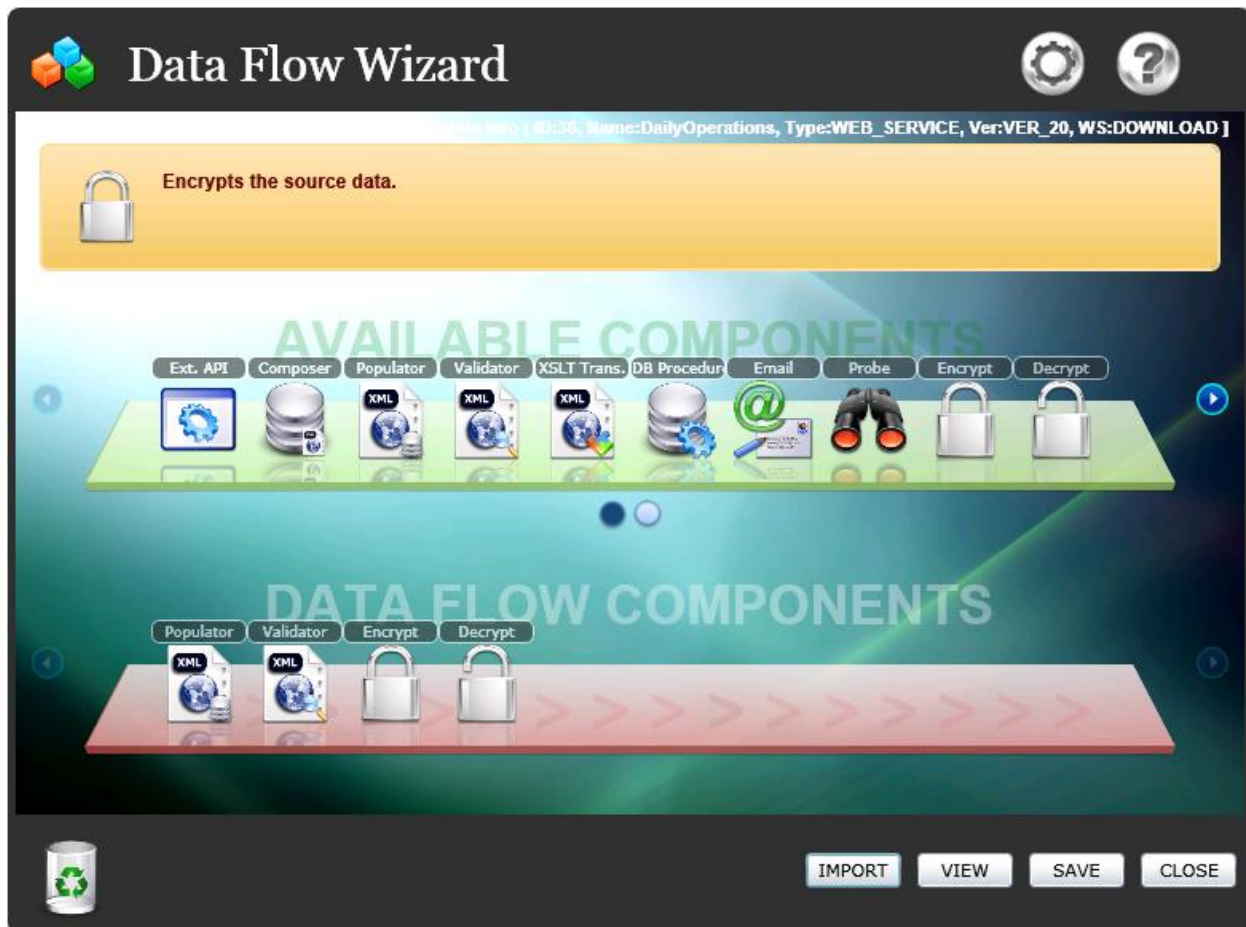
The changes made to the Operation can be saved by clicking on the  button.

### 2.3 Creating Flows from the Data Flow Wizard

To add a data flow to the Node Operation using the Data Flow Wizard, you will need to identify which data flow components shall be executed for the appropriate data flow. A data flow component is a piece or process of a data flow that can be called to perform a number of activities. The components available for addition are as follows:

- **External API:** Calls an external API to trigger the data flow
  - **Composer:** Calls the composer library to generate an XML file using data retrieved from a database
  - **Populator:** Calls the Populator library to populate a database using data from an XML file
  - **Validator:** Calls the Validator library to validate an XML instance
  - **XSLT Transformer:** Calls the transformer library to transform an XML file according to a specified XSLT style sheet
  - **DB Procedure:** Executes a stored procedure
  - **Email:** Sends email
  - **Probe:** Saves the data flow information to the database
  - **Encrypt:** Encrypts the source data
  - **Decrypt:** Decrypts the source data
  - **Zip:** Compresses the source data into a WINZIP file
  - **Unzip:** Decompresses the compressed source file
  - **WebServices:** Invokes a web service by the specified method
-

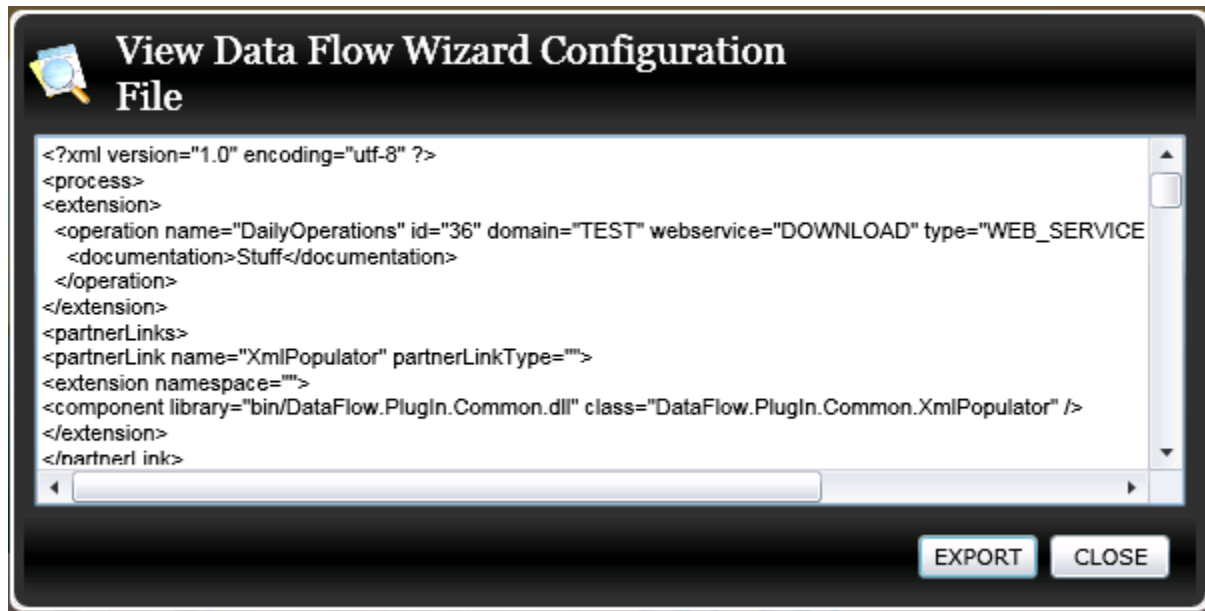
- **SaveToNode:** Saves the previous output to the Node database
- **GetStatus:** Gets the status of a submitted file and downloads any error/processing report returned by the CDX for the transaction



**Figure 2-9 Data Flow Wizard**

The data flow components can be added by clicking on the appropriate icon. This action will populate tray 2 with the selected component, indication selection and association to the data flow.

The changes made to the Data Flow can be saved by clicking on the **SAVE** button. Clicking on the **VIEW** button will display the Data Flow Configuration file, which can be exported as an XML file by clicking on the **EXPORT** button:



*Figure 2-10 Data Flow Wizard Configuration File*

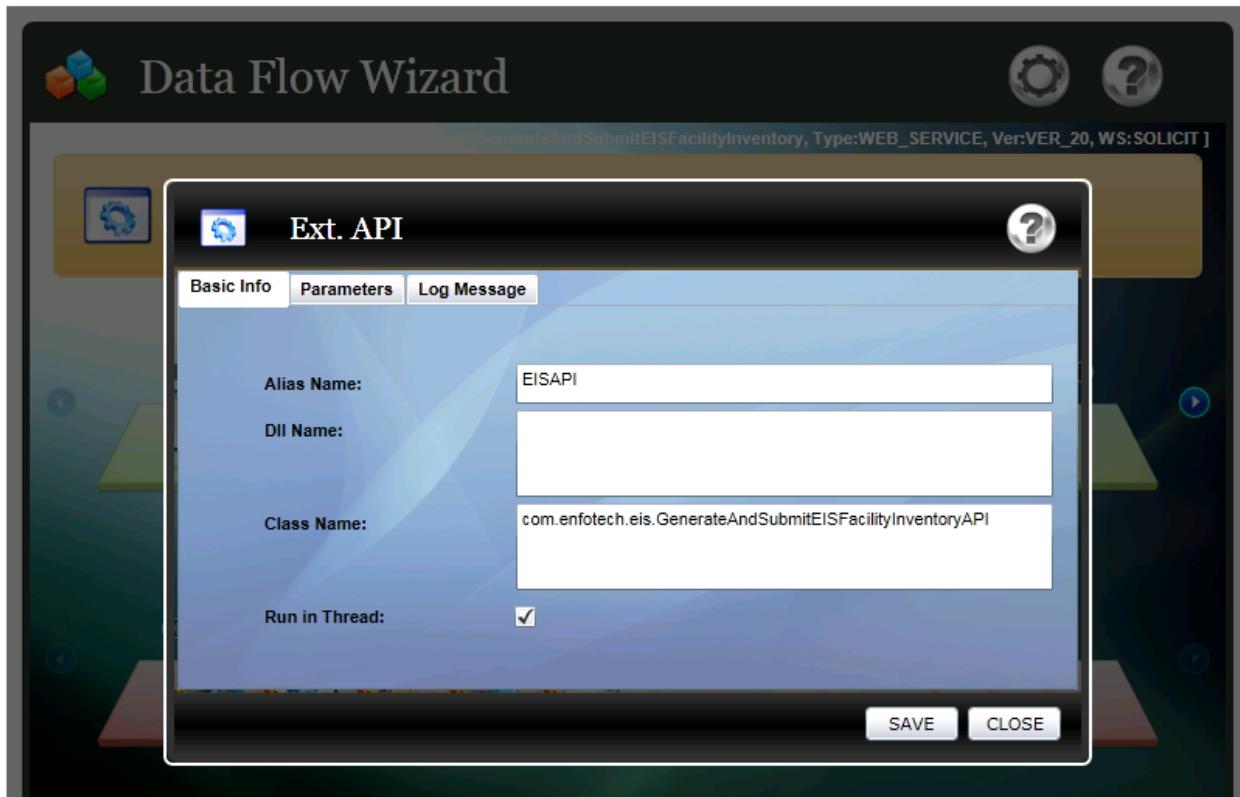
Data flow components can be removed from the Data Flow by dragging the appropriate icon from tray 2 into the recycle bin.

### 2.3.1 External API Data Flow Component

To use the External API Data Flow Component, select on the *External API* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the External API icon and provide the following information under the Basic Info tab:

- **Alias Name:** Identifier for the External API component
- **DLL Name:** Identifies the name of the DLL created to plug-into the data flow, It is useless for java version.
- **Class Name:** Identifies which class within the implemented DLL to reference for the data flow
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User

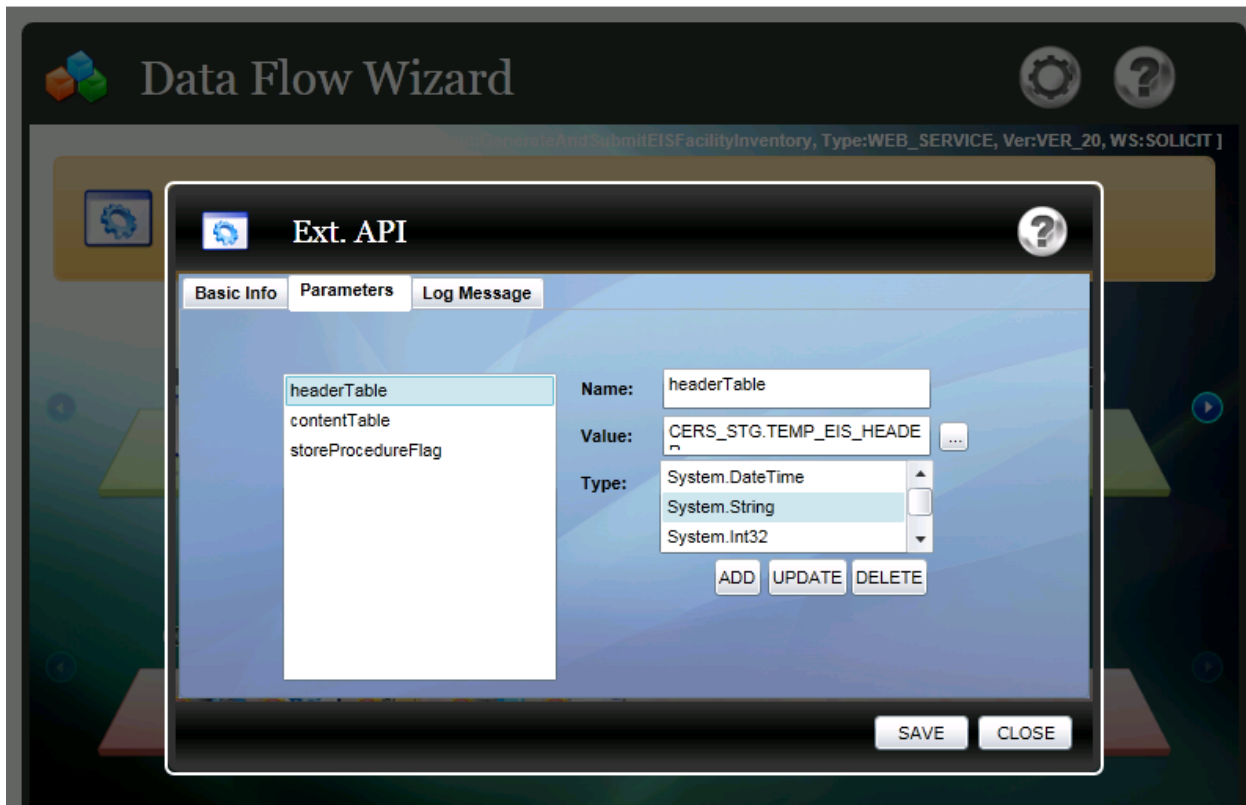




Select the Save button to save the properties for the External API data flow component.

Proceed to the Parameters tab of the External API component. Under this tab, any input parameters used within the DLL can be defined here. To add new parameters, select on the Add button and enter the following information:

- **Name:** Identifier for the parameter
- **Value:** Value associated with the parameter, to pass into the DLL
- **Type:** Defines the data type of the parameter

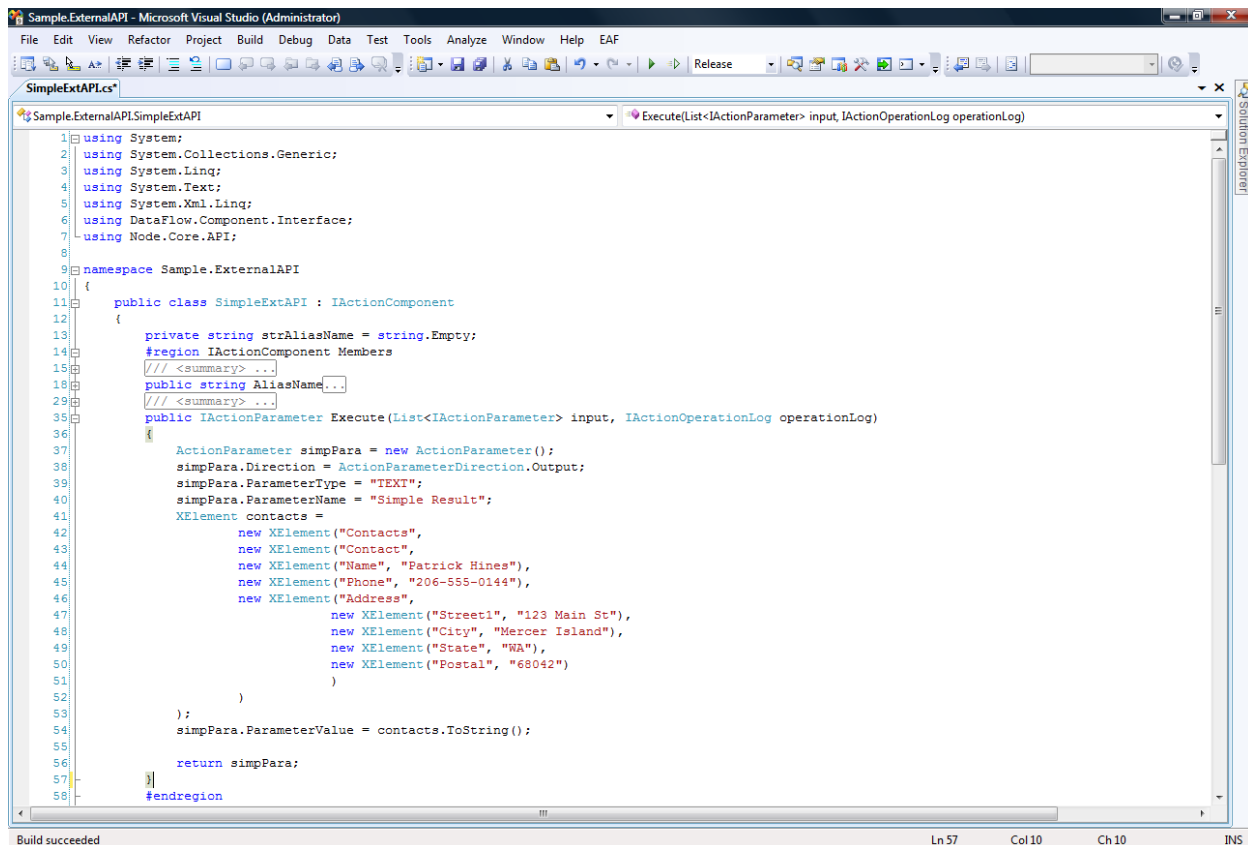


Use the Update and Delete buttons to update information for the new parameter or delete an existing parameter from the list.

Once all the information for the External API component has been provided, select on the Save button to save the properties for the External API data flow component. Select on the Close button to close the window.

The following diagram shows a class that could be used to the External API component. The class can be placed anywhere in Application server which can be accessed. It is recommended to place in the Node.TaskService/Node.TaskHandler directory if the component is used by schedule task or in the Node.WebServices/bin directory if component is used by web service.

- The class has to implement DataFlow.Component.Interface.IActionComponent.
- The Execute method is the entry point of the component with two input parameters: input and operationlog. The input contains parameters which are defined during the component creation. The operationlog stored the current operation related information such as transaction id, security token...etc.
- The class shown in following diagram constructs a simple xml file as output.



### 2.3.2 Composer Data Flow Component

To use the Composer Data Flow Component, select on the *Composer* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Composer icon and provide the following information under the Basic Info tab:

- **Alias Name:** Identifier for the Composer component
- **Database Type:** Defines the type of the source database
- **Connection String:** Defines the server, database, username and password information to access the source database.
- **Mapper:** Lists all Composer mappers available in the system for use. Mappers are the XML files that contain the Database to XML mapping information. Highlight selection of the desired Composer mapper will indicate the appropriate mapper to use for the data component.
- **Template:** Lists all Composer templates available in the system for use. Templates are the XML skeleton files that contain the XML schema information. Highlight selection of the desired Composer Template will indicate the appropriate template to use for the data component.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User

The screenshot shows the 'Composer' dialog box. At the top, there's a title bar with a database icon on the left and a question mark icon on the right. Below the title bar are three tabs: 'Basic Info', 'Parameters', and 'Log Message'. The 'Basic Info' tab is selected. The main area of the dialog has a blue background with a subtle pattern. It contains the following elements:

- Alias Name:** A text input field.
- Database Type:** Three radio buttons labeled 'MSSQL', 'Oracle', and 'Other'.
- Connection String:** A large text input field.
- TEST:** A button located to the right of the 'Connection String' field.
- Mapper:** A text input field with a browse button (three dots) to its right.
- Template:** A text input field with a browse button (three dots) to its right.
- Run in Thread:** A checkbox.

Select the Save button to save the properties for Composer data flow component.

Proceed to the Parameters tab of the Composer component. Under this tab, any input parameters used by the component mapper can be defined here. To add new parameters, select on the Add button and enter the following information:

- **Name:** Identifier for the parameter
- **Value:** Value associated with the parameter, to pass into the composer
- **Type:** Defines the data type of the parameter

Use the Update and Delete buttons to update information for the new parameter or delete an existing parameter from the list.

Once all the information for the Composer component has been defined, select on the Save button to save the properties for the Composer data flow component. Select on the Close button to close the window.

---

### 2.3.3 Populator Data Flow Component

To use the Populator Data Flow Component, select on the *Populator* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Populator icon and provide the following information:

- **Alias Name:** Identifier for the Populator component
- **Database Type:** Defines the type of the destination database
- **Connection String:** Defines the server, database, username and password information to access the destination database.
- **Mapper:** Lists all Populator mappers available in the system for use. Mappers are the XML files that contain the XML to Database mapping information. Highlight selection of the desired populator mapper will indicate the appropriate mapper to use for the data component.
- **Source:** Indicates what instance file to use. If other data flow components are used prior to the Populator Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User

The screenshot shows the 'Populator' configuration window with a dark header and a light blue body. The header contains an XML icon, the title 'Populator', and a help icon. Below the header are three tabs: 'Basic Info' (selected), 'Parameters', and 'Log Message'. The 'Basic Info' tab contains the following fields and controls:

- Alias Name:** A text input field.
- Database Type:** Three radio buttons labeled 'MSSQL', 'Oracle', and 'Other'.
- Connection String:** A large text input field with a 'TEST' button to its right.
- Mapper:** A text input field with a dropdown arrow button to its right.
- Source:** A text input field with an 'ADD PARAMETER' button above it.
- Run in Thread:** A checkbox.

At the bottom right of the window are 'SAVE' and 'CLOSE' buttons.

Select the Save button to save the properties for Populator data flow component.

---

Proceed to the Parameters tab of the Populator component. Under this tab, any input parameters used by the component mapper can be defined here. To add new parameters, select on the Add button and enter the following information:

- **Name:** Identifier for the parameter
- **Value:** Value associated with the parameter, to pass into the populator
- **Type:** Defines the data type of the parameter

Use the Update and Delete buttons to update information for the new parameter or delete an existing parameter from the list.

Once all the information for the Populator component has been defined, select on the Save button to save the properties for the Populator data flow component. Select on the Close button to close the window.

### 2.3.4 Validator Data Flow Component

To use the Validator Data Flow Component, select on the *Validator* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Validator icon and provide the following information:

- **Alias Name:** Identifier for the Validator component
  - **Database Type:** Defines the type of the source/destination database
  - **Connection String:** Defines the server, database, username and password information to access the source/destination database.
  - **Rule:** Lists all Validator Rule files available in the system for use. Rules are the XML files that contain the validation rules for generating an XML file or parsing an XML file. Highlight selection of the desired validator rule will indicate the appropriate rule to use for the data component.
  - **Source:** Indicates what instance file to use. If other data flow components are used prior to the Validator Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
  - **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User
-

The image shows a 'Validator' dialog box with a dark title bar and a light blue background. It has two tabs: 'Basic Info' (selected) and 'Log Message'. The 'Basic Info' tab contains several fields: 'Alias Name' with the text 'MyValidate', 'Database Type' with radio buttons for 'MSSQL' (selected), 'Oracle', and 'Other', 'Connection String' with a large empty text area, 'Rule' with a large empty text area and a small menu icon to its right, 'Source' with a 'REMOVE PARAMETER' button and a text field containing '{mytestqueryzip::Output}', and 'Run in Thread' with an unchecked checkbox. A 'TEST' button is located to the right of the 'Connection String' field. At the bottom right are 'SAVE' and 'CLOSE' buttons. A help icon (?) is in the top right corner of the title bar.

Once all the information for the Validator component has been defined, select on the Save button to save the properties for the Validator data flow component. Select on the Close button to close the window.

### 2.3.5 XSLT Transformer Data Flow Component

To use the XSLT Data Flow Component, select on the *XSLT* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the XSLT icon and provide the following information:

- **Alias Name:** Identifier for the XSLT component
  - **XSLT:** Lists all style sheets available in the system for use. Style sheets are logic files that contain the logic for transforming data into a particular format for easier viewing. Highlight selection of the desired style sheet will indicate the appropriate rule to use for the data component.
  - **Source:** Indicates what instance file to use. If other data flow components are used prior to the XSLT Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
  - **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User
-

The screenshot shows the 'XSLT Trans.' configuration window. The 'Basic Info' tab is active, displaying fields for 'Alias Name' (MyxsltTrans), 'XSLT' (a list of files), 'Source' (with an 'ADD PARAMETER' button), and 'Run in Thread' (unchecked). The 'Log Message' tab is also visible. The window has a title bar with an XML icon and a help icon. At the bottom right are 'SAVE' and 'CLOSE' buttons.

Once all the information for the XSLT component has been defined, select on the Save button to save the properties for the XSLT data flow component. Select on the Close button to close the window.

### 2.3.6 DB Procedure Data Flow Component

To use the DB Procedure Data Flow Component, select on the *DB Procedure* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the DB Procedure icon and provide the following information:

- **Alias Name:** Identifier for the DB procedure component
- **Database Type:** Defines the type of the database
- **Connection String:** Defines the server, database, username and password information to access the database.
- **SP Name:** Identifies the name of the stored procedure to use, as created in the database
- **SP Time Out (in seconds):** Identifies the length of execution of a stored procedure before the system errors out
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User



**DB Procedure**

Basic Info Parameters Log Message

Alias Name: EISStoreProcedure

Database Type: ☐ MSSQL ☒ Oracle ☐ Other

Connection String: node

TEST

SP Name: CERS\_STG.USB\_GENERATE\_CERS\_XML

SP Time Out (in Seconds):

Run in Thread: ☒

SAVE CLOSE

Select the Save button to save the properties for DB Procedure data flow component.

Proceed to the Parameters tab of the DB Procedure component. Under this tab, any input parameters used by the db procedure can be defined here. To add new parameters, select on the Add button and enter the following information:

- **Name:** Identifier for the parameter
- **Value:** Value associated with the parameter, to pass into the db procedure
- **Type:** Defines the data type of the parameter

Use the Update and Delete buttons to update information for the new parameter or delete an existing parameter from the list.

Once all the information for the DB Procedure component has been defined, select on the Save button to save the properties for the DB Procedure data flow component. Select on the Close button to close the window.

### 2.3.7 Email Data Flow Component

To use the Email Data Flow Component, select on the *Email* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Email icon and provide the following information:

- **Alias Name:** Identifier for the XSLT component
-

- **To List:** Defines the email addresses of all email recipients (multiple emails can be specified and should be separated by a ',')
- **Subject:** Identifies the subject of the email
- **Content:** Identifies the content of the email (email body)
- **Attachments:** Indicates what attachments to attach with the email. If other data flow components are used prior to the XSLT Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User

The screenshot shows a window titled "Email" with a dark header bar containing an email icon on the left and a help icon on the right. Below the header, there are two tabs: "Basic Info" (selected) and "Log Message". The main area of the window is light blue and contains several input fields and buttons. The "Alias Name:" field contains "EISSendEmail". The "To List:" field contains "myemail@enfotech.com" and has a small "..." button to its right. The "Subject:" field contains "eis test". The "Content:" field is a large text area containing "test". Below these fields, there are two buttons: "INSERT PARAMETER" and "ADD PARAMETER". The "Attachment:" label is positioned to the left of the "ADD PARAMETER" button. Below the "Attachment:" label, there is a large empty rectangular box. The "Run in Thread:" checkbox is checked. At the bottom right of the window, there are two buttons: "SAVE" and "CLOSE".

Once all the information for the Email component has been defined, select on the Save button to save the properties for the Email data flow component. Select on the Close button to close the window.

### 2.3.8 Probe Data Flow Component

To use the Probe Data Flow Component, select on the *Probe* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Probe icon and provide the following information:

---

- **Alias Name:** Identifier for the Probe component
- **Transaction ID:** Indicates whether or not to save the Transaction ID to the database
- **Data Flow:** Indicates whether or not to save the Data Flow name to the database
- **Request IP:** Indicates whether or not to save the Requestor's IP to the database
- **Source:** Indicates what files to save to the database. If other data flow components are used prior to the Email Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User



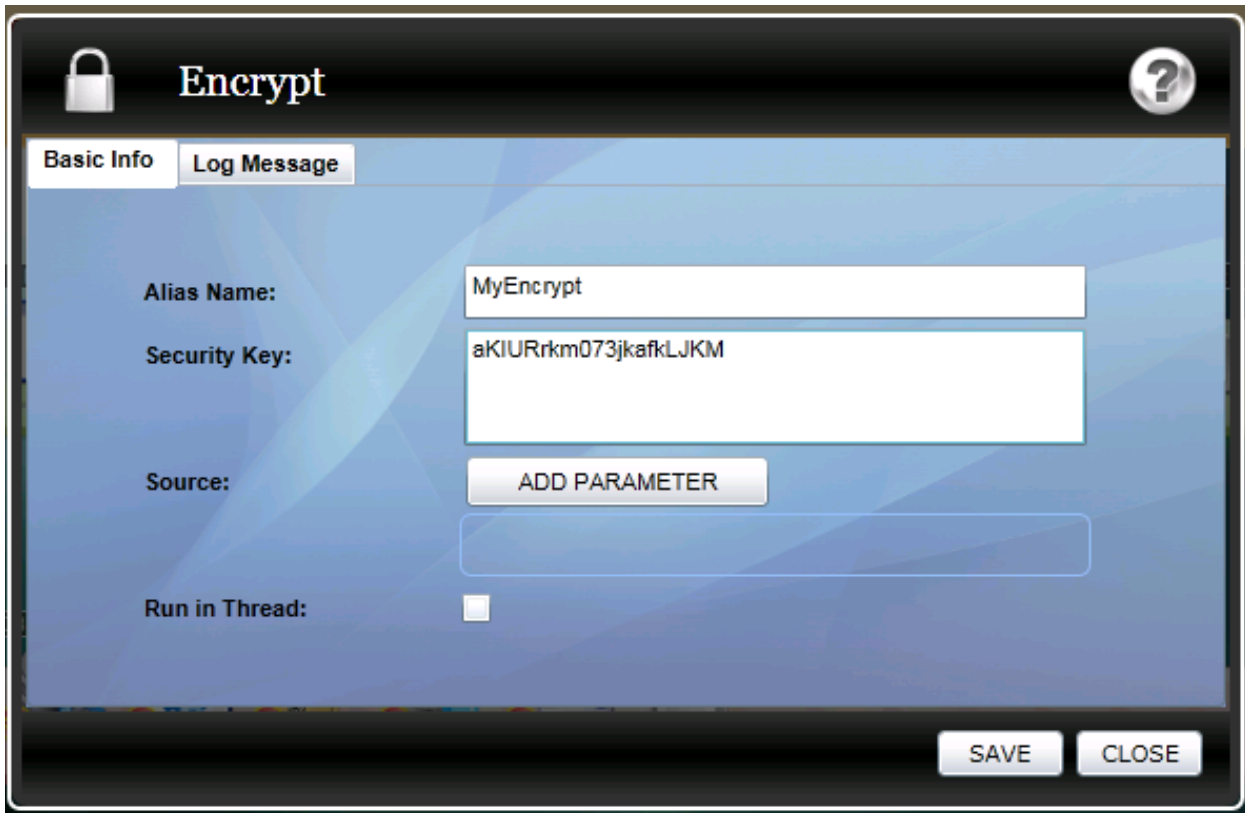
Once all the information for the Email component has been defined, select on the Save button to save the properties for the Email data flow component. Select on the Close button to close the window.

### 2.3.9 Encrypt Data Flow Component

To use the Encrypt Data Flow Component, select on the *Encrypt* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Encrypt icon and provide the following information:

- **Alias Name:** Identifier for the Encrypt component
  - **Security Key:** Identifies the security key to use to encrypt the source file
  - **Source:** Indicates what files to encrypt. If other data flow components are used prior to the Encrypt Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
-

- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User

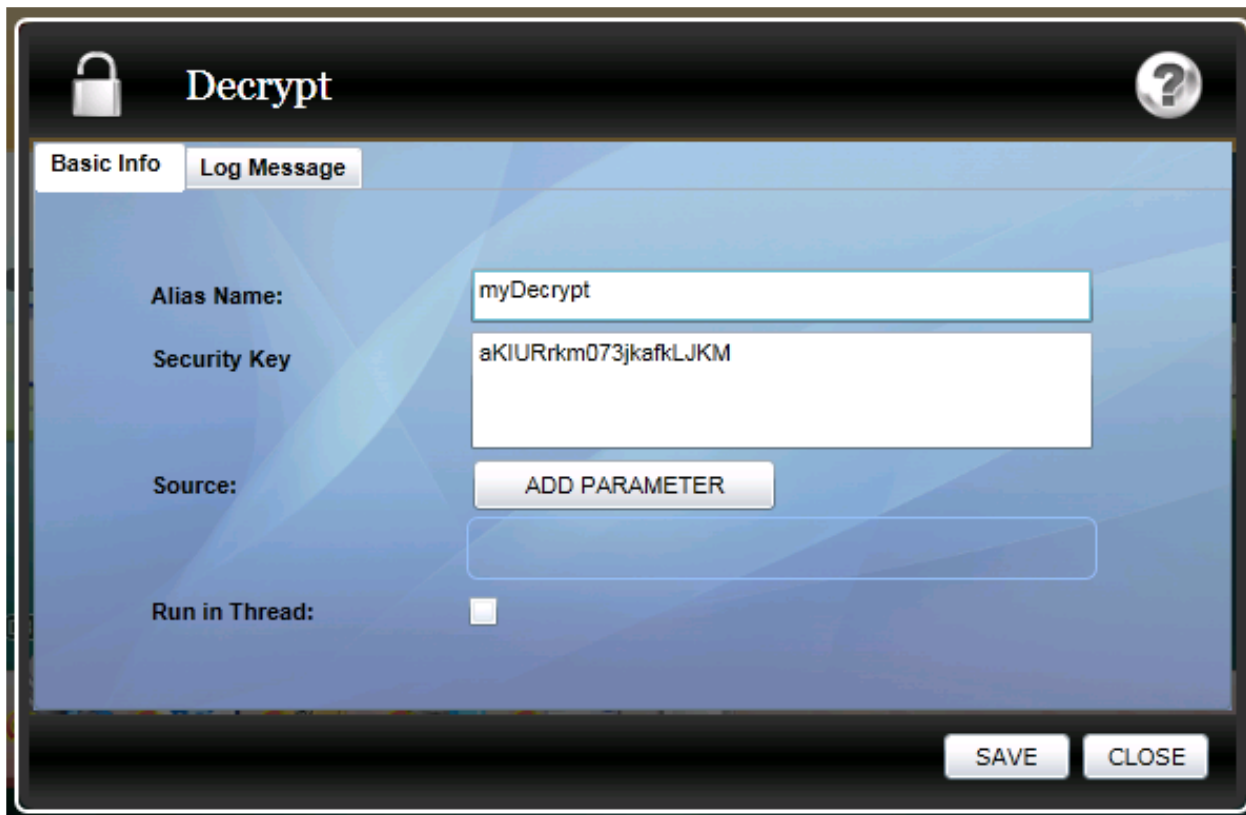


Once all the information for the Encrypt component has been defined, select on the Save button to save the properties for the Encrypt data flow component. Select on the Close button to close the window.

### 2.3.10 Decrypt Data Flow Component

To use the Decrypt Data Flow Component, select on the *Decrypt* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Decrypt icon and provide the following information:

- **Alias Name:** Identifier for the Decrypt component
  - **Security Key:** Identifies the security key to use to decrypt the source file
  - **Source:** Indicates what files to decrypt. If other data flow components are used prior to the Decrypt Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
  - **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User
-



The screenshot shows a 'Decrypt' configuration window. It has a title bar with a lock icon and the title 'Decrypt'. Below the title bar are two tabs: 'Basic Info' and 'Log Message'. The 'Basic Info' tab is active and contains the following fields:

- Alias Name:** A text box containing 'myDecrypt'.
- Security Key:** A text box containing 'aKIURrkm073jkafkLJKM'.
- Source:** A text box with an 'ADD PARAMETER' button above it and an empty text box below it.
- Run in Thread:** A checkbox that is currently unchecked.

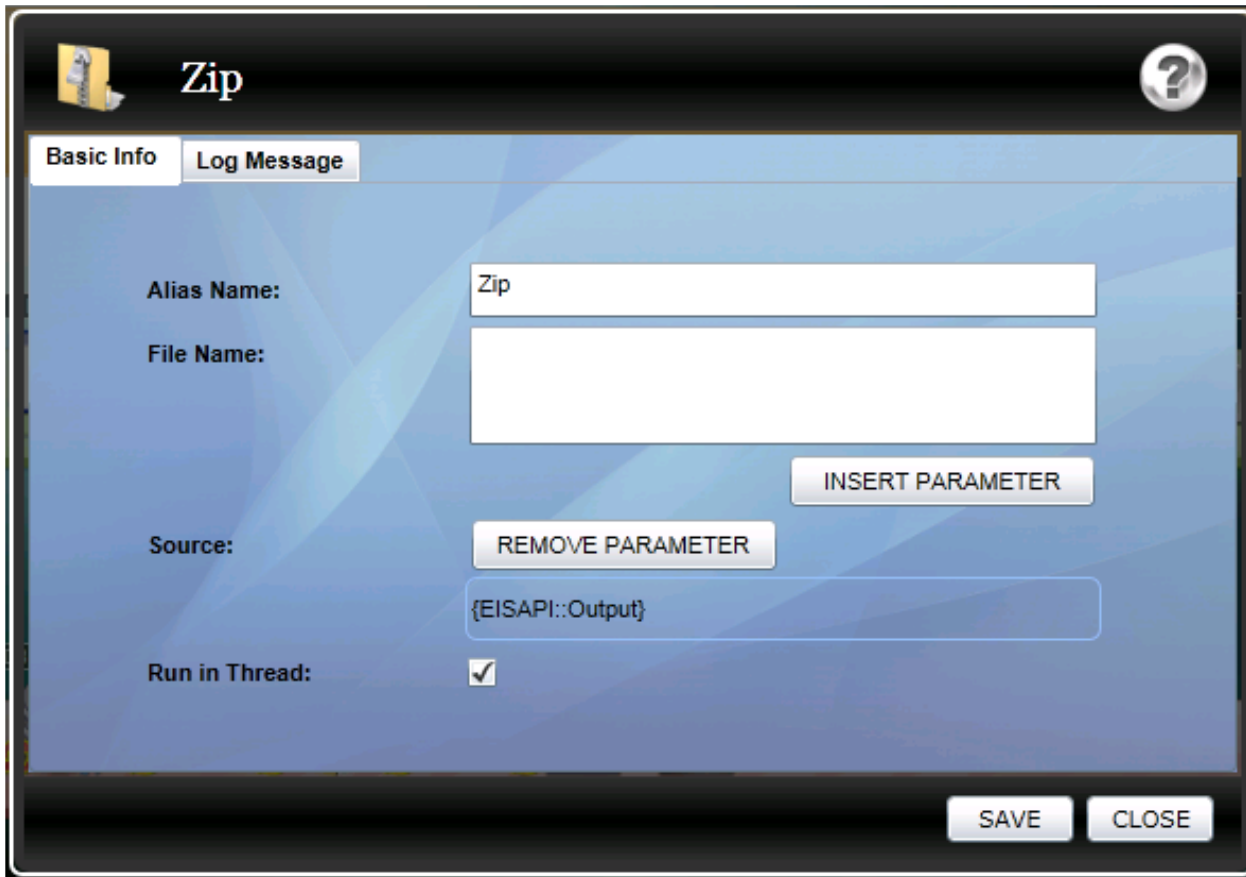
At the bottom right of the window are two buttons: 'SAVE' and 'CLOSE'.

Once all the information for the Decrypt component has been defined, select on the Save button to save the properties for the Decrypt data flow component. Select on the Close button to close the window.

### 2.3.11 Zip Data Flow Component

To use the Zip Data Flow Component, select on the *Zip* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Zip icon and provide the following information:

- **Alias Name:** Identifier for the Zip component
- **File Name:** Identifies the name to assign to the file after compression
- **Source:** Indicates what files to compress. If other data flow components are used prior to the Zip Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User



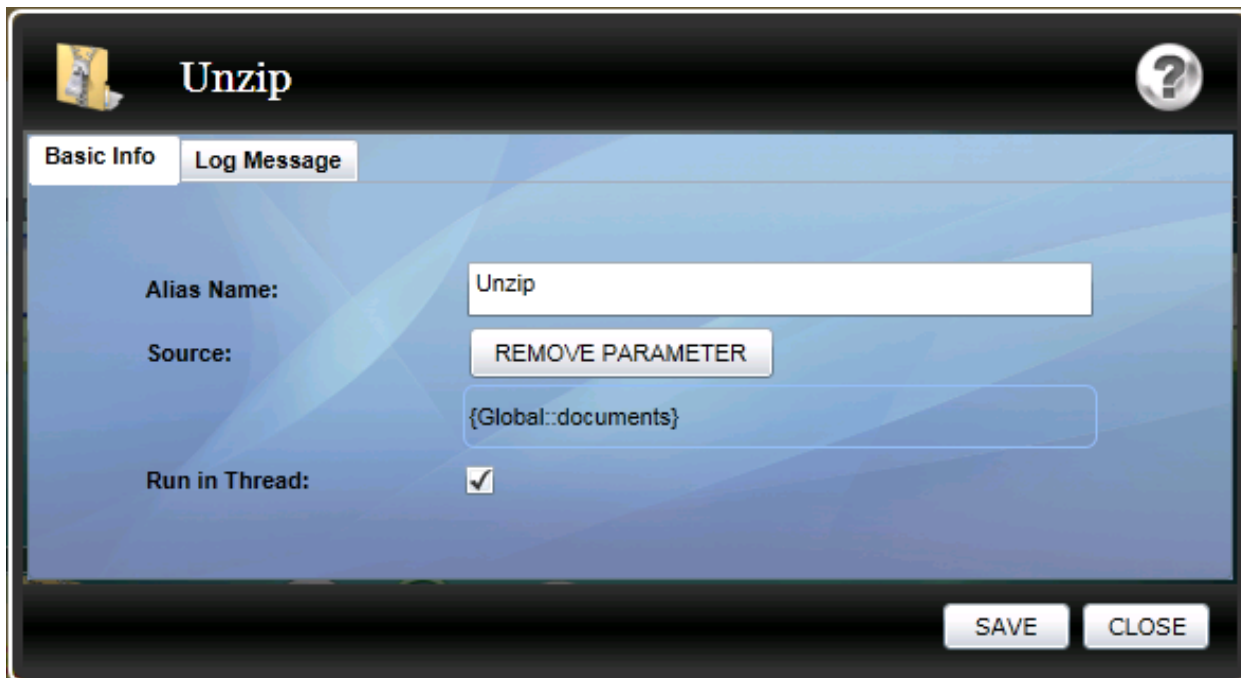
Once

all the information for the Zip component has been defined, select on the Save button to save the properties for the Zip data flow component. Select on the Close button to close the window.

### 2.3.12 Unzip Data Flow Component

To use the Unzip Data Flow Component, select on the *Unzip* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Unzip icon and provide the following information:

- **Alias Name:** Identifier for the Unzip component
- **Source:** Indicates what files to decompress. If other data flow components are used prior to the Zip Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User

The screenshot shows a configuration window for the 'Unzip' component. The window has a title bar with a folder icon, the text 'Unzip', and a help icon. Below the title bar are two tabs: 'Basic Info' (selected) and 'Log Message'. The 'Basic Info' tab contains three fields: 'Alias Name' with the value 'Unzip', 'Source' with a 'REMOVE PARAMETER' button and the value '{Global::documents}', and 'Run in Thread' with a checked checkbox. At the bottom right are 'SAVE' and 'CLOSE' buttons.

Once all the information for the Unzip component has been defined, select on the Save button to save the properties for the Unzip data flow component. Select on the Close button to close the window.

### 2.3.13 Web Services Data Flow Component

To use the WebServices Data Flow Component, select on the *WebServices* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the WebServices icon and provide the following information:

- **Alias Name:** Identifier for the WebServices component
- **Node URL:** Identifies the Node address to invoke the webservice for
- **Node Version:** Identifies the Node specification the specified address is for
- **Service Type:** Identifies the type of service call to make to the specified Node
- **Domain:** Identifies the Authentication Domain to authenticate to
- **User Name:** Identifies the user name approved by the destination Node to authenticate with
- **Password:** Identifies the password corresponding to the User Name to authenticate with
- **Auth. Method:** Defines the authentication method to use to authenticate with
- **Source:** Indicates what parameters to submit to the destination Node. If other data flow components are used prior to the WebServices Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::document} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User



The image shows a 'WebServices' configuration dialog box. It has a title bar with a globe icon on the left and a help icon on the right. Below the title bar are three tabs: 'Basic Info', 'Parameters', and 'Log Message'. The 'Basic Info' tab is selected. The form contains the following fields and controls:

- Alias Name:** Text box containing 'mytestwizardquery'.
- Node URL:** Text box containing 'http://server1:8080/Node.WebServices/services/NetworkNode2'.
- Node Version:** Radio buttons for 'VER\_11' and 'VER\_20'. 'VER\_20' is selected.
- Service Type:** Radio buttons for 'Submit' and 'Query'. 'Query' is selected.
- Domain:** Text box containing 'Test'.
- User Name:** Text box containing 'DCMLocal'.
- Password:** Text box containing 'password'.
- Auth. Method:** Text box containing 'PASSWORD'.
- Source:** A button labeled 'ADD PARAMETER' above an empty text box.
- Run in Thread:** A checkbox that is currently unchecked.

At the bottom right of the dialog are two buttons: 'SAVE' and 'CLOSE'.

Select the Save button to save the properties for WebServices data flow component.

Proceed to the Parameters tab of the WebServices component. Under this tab, any input parameters used by the web service can be defined here. To add new parameters, select on the Add button and enter the following information:

- **Name:** Identifier for the parameter
- **Value:** Value associated with the parameter, to pass into the db procedure
- **Type:** Defines the data type of the parameter

Use the Update and Delete buttons to update information for the new parameter or delete an existing parameter from the list.

Once all the information for the WebServices component has been defined, select on the Save button to save the properties for the WebServices data flow component. Select on the Close button to close the window.

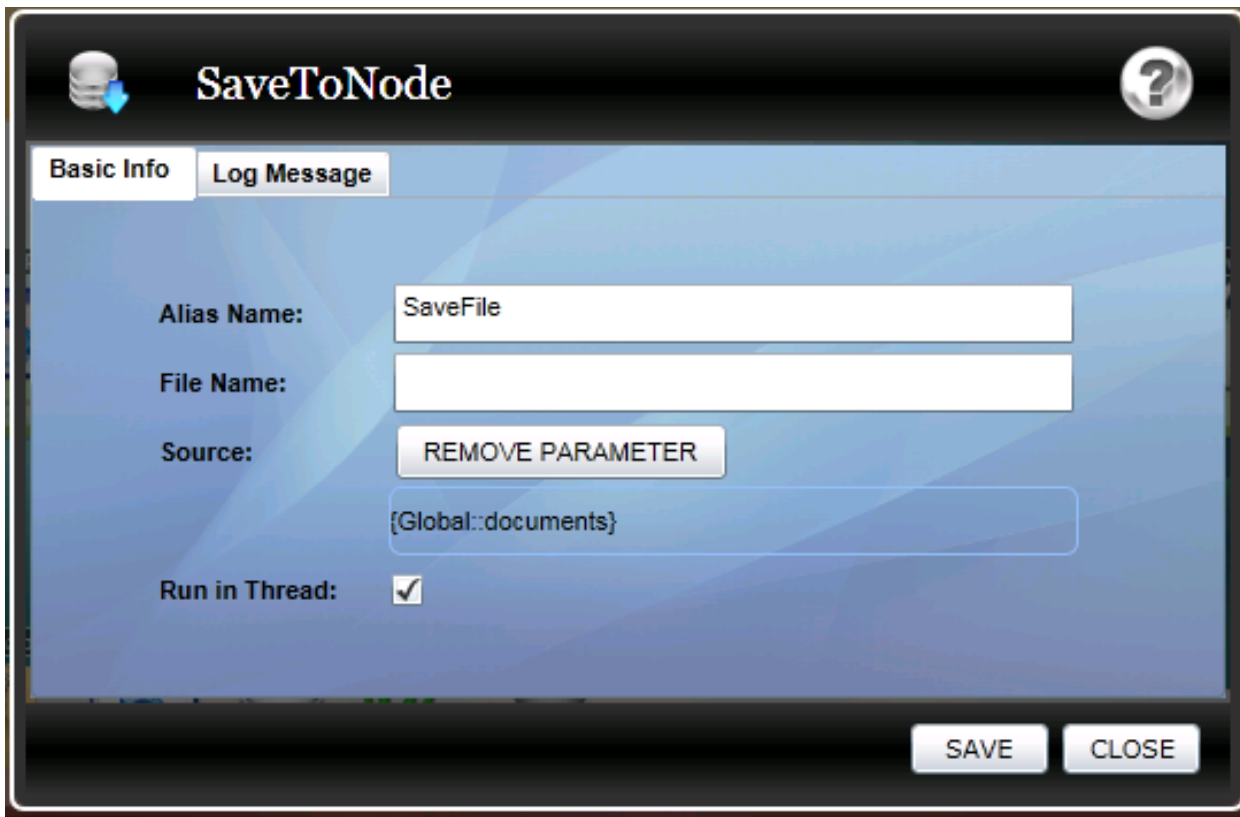
---



### 2.3.14 Save to Node Data Flow Component

To use the Save to Node Data Flow Component, select on the *Save to Node* icon from tray 1, which will populate tray 2 and associate the component to the data flow. From tray 2, double click on the Save to Node icon and provide the following information:

- **Alias Name:** Identifier for the Save to Node component
- **File Name:** Identifies the name to associate with the file when saved to the database
- **Source:** Indicates what files to save to the Node. If other data flow components are used prior to the Save to Node Data Flow component, the Source may be the outcome of the appropriate component, otherwise, the {Global::documents} parameter should be used.
- **Run in Thread:** Indicates whether or not to run the data flow in threaded mode (background process) after it has been initiated by the User



Once all the information for the Save to Node component has been defined, select on the Save button to save the properties for the Save to Node data flow component. Select on the Close button to close the window.

## 2.4 Writing Custom Web Service or Task Service Class

### 2.4.1 .Net Version:

In order to write custom code that you want to plug into the Node, you will need to reference the following when developing your code:

- Node.Core.dll
  - Node.Core2.dll
  - DataFlow.Component.dll
-

1. Write .NET Class that implements logic for the new flow.  
*(Note: You may have multiple classes for each flow, for non-Wizard flows, this would correspond to (1) one or more Preprocesses, (2) a Process, and (3) one or more post-processes)*
    - a. Web Service Operation:
      - i. Create a new class
        1. Implement the `Node.Core.Biz.Interfaces.WebServiceName.IPreProcess` or `Node.Core.Biz.Interfaces.WebServiceName.IProcess` or `Node.Core.Biz.Interfaces.WebServiceName.IPostProcess` interface depending upon whether the class is for a pre-process, process or a post-process.
        2. Define the method `Execute` in the interface you are implementing. The method has input arguments for custom defined objects: `PreParam`, `ProcParam`, or `PostParam`. These custom objects contain functions for accessing values available to a class implementing a pre-process, process or post-process. Values can be added to the `Hashtable` contained in these custom objects in order to reference any of the following pre/post-process or process.
        3. These custom objects refer to methods that allow access to specific details of the operation.
      - b. Scheduled Task Operation:
        - i. Create a new class within the same Package
          1. Implement the `Node.Core.Biz.Interfaces.Task.IProcess` interface.
          2. Define the method `Execute` in the interface you are implementing. The method has input arguments for custom defined objects: string array and `ProcParam`. The string array contains parameter values which are defined during Operation Creation. The `ProcParam` stores the logic of the process that the handler reads
  2. Plug the new data flow into the Node
    - a. Create Operations for the Domain
      - i. Create pre-process, process, and post-process for the Operation if the Operation is a Web Service Operation or create a class and a method for the class if the Operation is a Scheduled Task.
      - ii. Copy the class/dll file into the bin directory under `Node.WebServices` if the operation is a `WebService` or the `Node.TaskHandler` directory if the operation is a Scheduled Task.
    - b. At the Domain Management GUI
      - i. Specify Web Services to be made available to the Domain
      - ii. Assign one or more Admin (Domain Admin) to the Domain
      - iii. Refer to the class(es) and the sequence of the parameters created in the previous step.
      - iv. Save the details on the page
-

### 2.4.1.1 Example Classes for Creating Operations:

#### Class for Web Service Operation:

The following figure shows a class that could be used for defining a process for “Solicit”. This dll needs to be stored under the Node.WebServices/bin directory.

- The class shown in the screenshot **implements the IProcess interface**.
- The class **calls the Execute Method** with the Security Token received from Authentication. ProcParam stores the logic of the process that the handler reads
- The Solicit method shown in the example writes a log message and uploads a document.

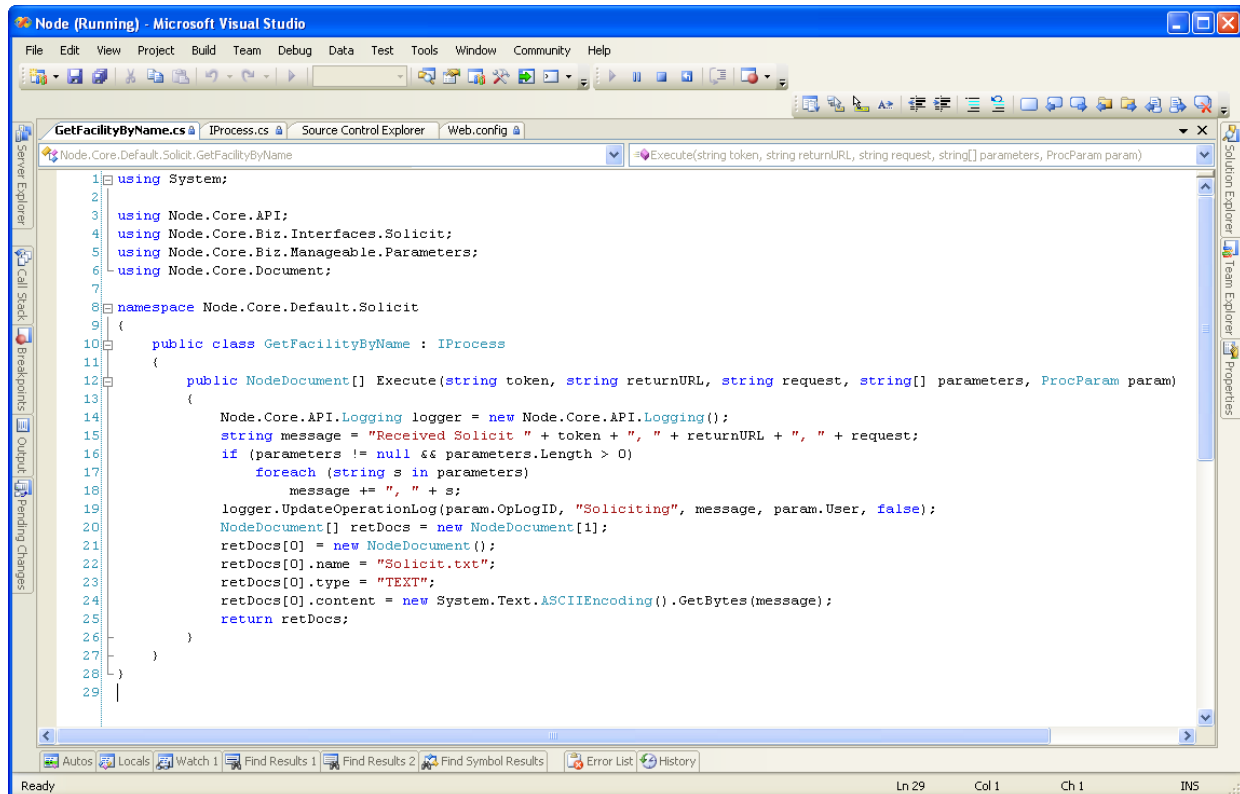
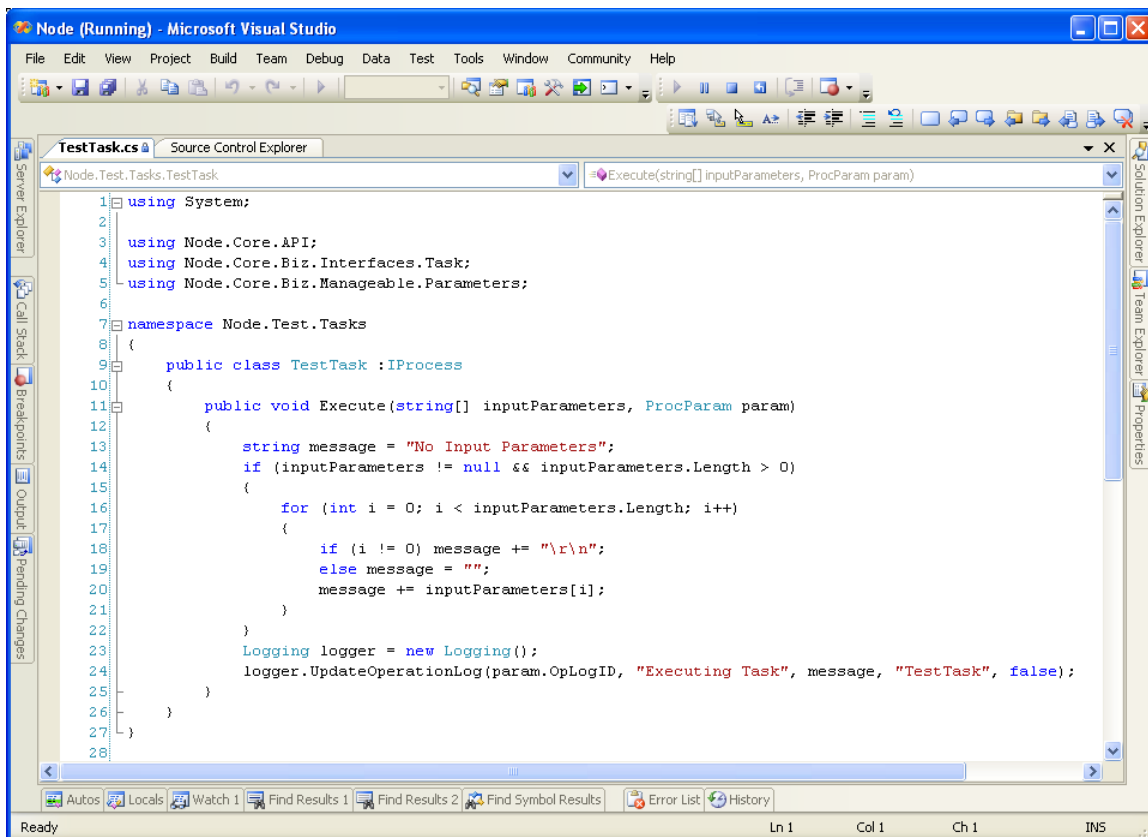


Figure 2-11 Class for Process

**Class for Scheduled Task Operation:**

The following diagram shows a class that could be used to schedule a task. This class needs to be stored under the Node.TaskService/Node.TaskHandler directory.

- The class shown in the screenshot **implements the IProcess interface**.
- The class **calls the Execute Method** with the **inputparameters** defined during operation creation. **param** stores the logic of the process that the handler reads.
- The schedule task shown in the diagram constructing message by looking through inputparameters and creating a log message.



**Figure 2-12 Class for Scheduled Task**

### 2.4.2 Java Version:

In order to write custom code that you want to plug into the Node, you will need to reference the following when developing your code:

- Node.jar
1. Write Java Class that implements logic for the new flow.  
(Note: You may have multiple classes for each flow, for non-Wizard flows, this would correspond to (1) one or more Preprocesses, (2) a Process, and (3) one or more post-processes)
    - a. Web Service Operation:
      - i. Create a new class
        1. Implement the  
`Node.Biz.Interfaces.WebServiceName.IPreProcess` or  
`Node.Biz.Interfaces.WebServiceName.IProcess` or  
`Node.Biz.Interfaces.WebServiceName.IPostProcess` interface depending upon whether the class is for a pre-process, process or a post-process.
        2. Define the method `Execute` in the interface you are implementing. The method has input arguments for custom defined objects: `PreParam`, `ProcParam`, or `PostParam`. These custom objects contain functions for accessing values available to a class implementing a pre-process, process or post-process. Values can be added to the `Hashtable` contained in these custom objects in order to reference any of the following pre/post-process or process.
        3. These custom objects refer to methods that allow access to specific details of the operation.
      - b. Scheduled Task Operation:
        - i. Create a new class within the same Package
          1. Implement the `Node.Biz.Interfaces.Task.IProcess` interface. Define the method `Execute` in the interface you are implementing. The method has input arguments for custom defined objects: `ProcParam`. These custom objects contain functions for accessing values available to a class implementing a pre-process, process or post-process. Values can be added to the `Hashtable` contained in these custom objects in order to reference any of the following pre/post-process or process.
          2. The class calls an empty constructor.
          3. Call the method(s) on the constructor to schedule the task
    2. Plug the new data flow into the Node
      - a. Create Operations for the Domain
        - i. Create pre-process, process, and post-process for the Operation if the Operation is a Web Service Operation or create a class and a method for the class if the Operation is a Scheduled Task.
        - ii. Compile and make a jar file for the new webservice or task classes, copy the jar file into the lib directory under `Node.WebServices/Web-INF` if the operation is a WebService or the lib directory under `Node.Task/Web-INF` directory if the operation is a Scheduled Task.
      - b. At the Domain Management GUI
        - i. Specify Web Services to be made available to the Domain
        - ii. Assign one or more Admin (Domain Admin) to the Domain
        - iii. Refer to the class(es) and the sequence of the parameters created in the previous step.
        - iv. Save the details on the page

### 2.4.2.1 Example Classes for Creating Operations:

#### Class for Web Service Operation:

The following figure shows a class that could be used for defining a process for “Solicit”. Please compile and make a jar file then put it under the Node.WebServices/Web-INF/lib directory.

- The class shown in the screenshot **implements the IProcess interface**.
- The class **calls the Execute Method** with the Security Token received from Authentication. ProcParam stores the logic of the process that the handler reads
- The Solicit method shown in the example writes a log message and uploads a document.

```
package Node.Biz.Default.Solicit;

import java.rmi.RemoteException;

import Node.API.NodeUtils;
import Node.Biz.Custom.ProcParam;
import Node.Biz.Interfaces.Solicit.IProcess;
import Node.Phrase;
import Node.WebServices.Document.ClsNodeDocument;
/**
 * <p>This class create Solicit Test Process.</p>
 * <p>Company: enfoTech & Consulting, Inc.</p>
 * @author enfoTech
 * @version 2.0
 */
public class Test implements IProcess {
    public Test() {

        public ClsNodeDocument Execute (String token, String returnUrl, String request,
Object[] params, ProcParam param) throws RemoteException
        {
            ClsNodeDocument doc = null;
            try {
                System.out.println("Hello World");
                NodeUtils utils = new NodeUtils();
                utils.UpdateOperationLog(param.GetLoggerName(), param.GetTransID(),
"Soliciting", "Token is: " + token + " returnUrl is: " + returnUrl, false);
                doc = new ClsNodeDocument();
                doc.setName("GetFacilityName.xml");
                doc.setType("XML");
                doc.setContent("<Result>This is my file content.</Result>".getBytes());
            } catch (Exception e) {
                throw new RemoteException(Phrase.InternalError,e);
            }
            return doc;
        }
    }
}
```

---

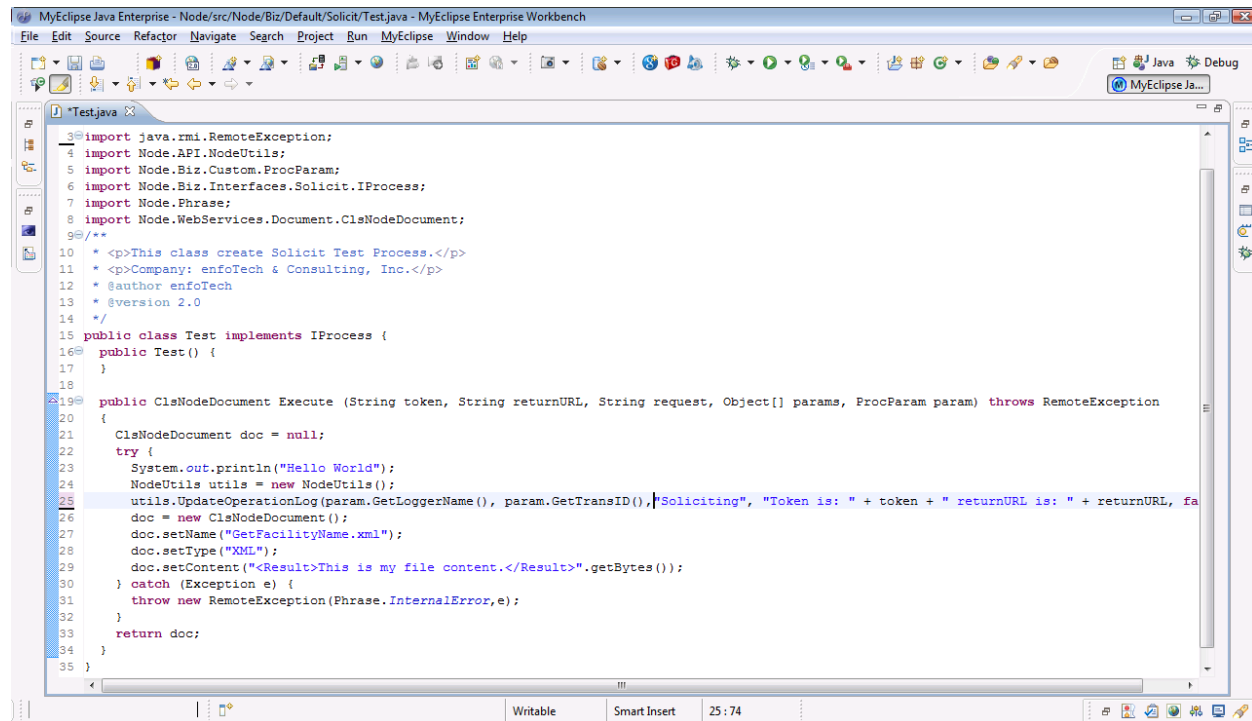


Figure 2-13 Class for Process

### Class for Scheduled Task Operation:

The following code and diagram shows a class that could be used to schedule a task. Please compile and make a jar file then put it under the Node.Task/Web-INF/lib directory.

For scheduling a task, no interfaces have been defined

- The class must have an empty constructor; TestTask in the example
- A method is then called (TestPlugIn) to schedule a task. In the example shown, the task sends a email and returns a string

```

package Node.Task.Test;

import java.io.StringWriter;
import java.io.PrintWriter;

import Node.API.Email;
import Node.API.NodeUtils;
import Node.Biz.Custom.ProcParam;
import Node.Biz.Interfaces.Task.IProcess;

public class TestTask implements IProcess {
    public TestTask() {
    }

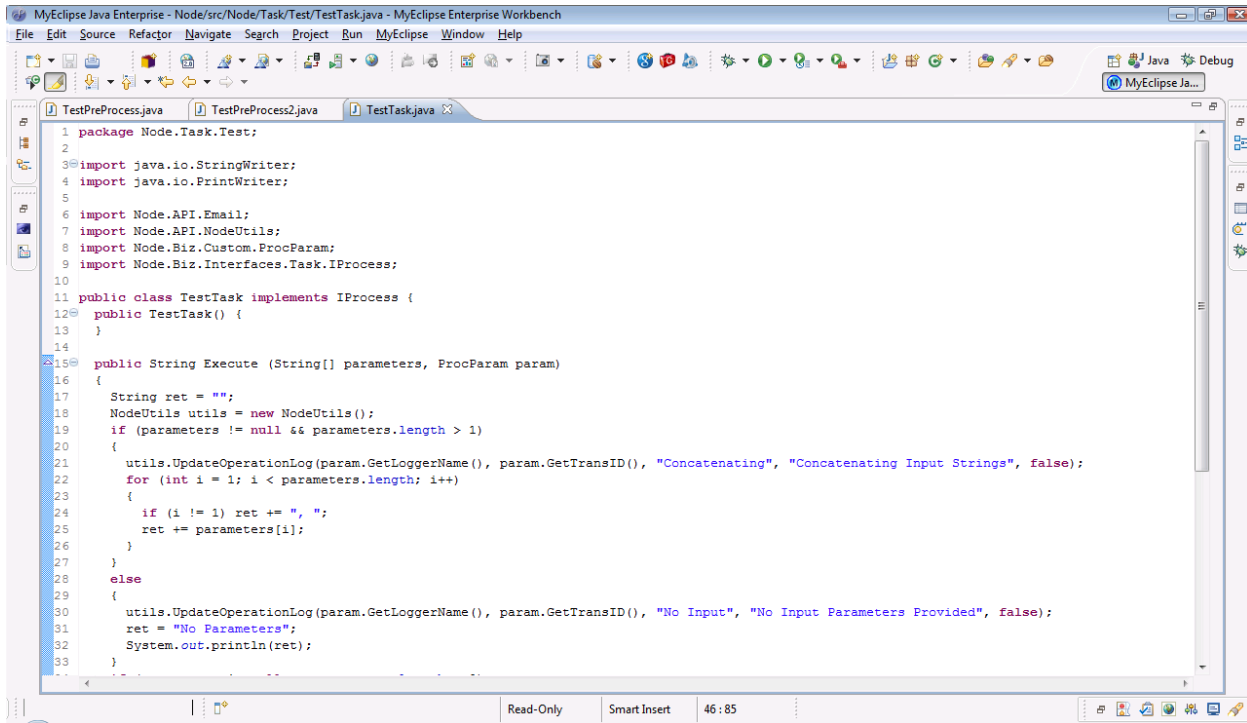
    public String Execute (String[] parameters, ProcParam param)
    {
        String ret = "";
        NodeUtils utils = new NodeUtils();

```

```
    if (parameters != null && parameters.length > 1)
    {
        utils.UpdateOperationLog(param.GetLoggerName(), param.GetTransID(),
"Concatenating", "Concatenating Input Strings", false);
        for (int i = 1; i < parameters.length; i++)
        {
            if (i != 1) ret += ", ";
            ret += parameters[i];
        }
    }
    else
    {
        utils.UpdateOperationLog(param.GetLoggerName(), param.GetTransID(), "No
Input", "No Input Parameters Provided", false);
        ret = "No Parameters";
        System.out.println(ret);
    }
    if (parameters != null && parameters.length > 0)
    {
        try
        {
            utils.UpdateOperationLog(param.GetLoggerName(), param.GetTransID(),
"Emailing", "Attempting to send Email", false);
            Email email = new Email(param.GetLoggerName());
            email.SendEmail("Test Task Status", ret, "nodetesttask@task.com",
parameters[0]);
        }
        catch (Exception e) {
            StringWriter sw = new StringWriter();
            e.printStackTrace(new PrintWriter(sw));
            try { sw.close(); } catch (Exception ex) { }
            utils.UpdateOperationLog(param.GetLoggerName(), param.GetTransID(), "Failed
Email", "Failed to send Email: " + sw.toString(), false);
        }
    }
    return ret;
}
```

---





**Figure 2-13 Class for Scheduled Task**

## 3 Application Program Interfaces (APIs)

### 3.1 .Net Version:

The following APIs are available for the Node:

- 1) Document Management API
- 2) Node Requestor API
- 3) API to update the current status of a request (i.e. Node Logging API)

Each API could be called by following two steps:

- 1) Call the constructor for the API
- 2) Call the methods defined in the API.

Some of the Node API's require a connection to the Node Database. If the test environment cannot connect to a test Node database, then comment out lines calling the DocumentManagement API or the Node Operation Logging API, as the API's only function correctly if there is a database connection established.

If the test environment does have access to a Node test database, specify the connection details in a file called App.config or Web.config, located in the classes directory of the application. The block should resemble this:

```
<connectionStrings>
    <add name="node" connectionString="Data
Source=ATHENDB.ENFOTECH.COM;User ID=XXXXX;Password=XXXXX"
providerName="System.Data.OracleClient"/>
</connectionStrings>
```

The name should be "node" and the connectionString should follow the standard connection string convention for Oracle or SQL Server. The providerName should System.Data.OracleClient if the Node database is an Oracle database, or System.Data.SqlClient if the Node database is a SQL Server database.

#### 1) Node Requestor API:

Node Requestor API utilizes a method for each Web Service.

- This API utilizes a constructor, Node.Core.API.NodeRequestor, a method for each Web Service (implemented via the constructor), and a class, NodeDocument, to access documents via the Submit and Download methods
- Instantiate the constructor. This object specifies the URL of the node to which the request is made
  - NodeRequestor requestor = new NodeRequestor("https://test.epacdxnode.net/cdx/services/NetworkNodePortType\_V10");
- NodePing()
  - Instantiate the built-in class String. Call the method NodePing() on the constructor with the initial argument as "hello"  
String ping = requestor.NodePing("hello");
- Authenticate()
  - Instantiate the built-in class String. Call the method Authenticate() on the constructor with the initial arguments as ("cdx", "test", "PASSWORD")  
String token = requestor.Authenticate("cdx", "test", "PASSWORD");
- Submit()
  - Instantiate the class Node.Core.Document.NodeDocument  
NodeDocument doc = new NodeDocument();

- Set the name, type, and content or Stream properties of an instantiated `Node.Core.Document.NodeDocument` class.  

```
NodeDocument doc = new NodeDocument();  
doc.name = "Test.xml";  
doc.type = "XML";  
doc.Stream = new MemoryStream();
```
- Call the Submit method.  

```
string transID = requestor.submit(token,null,null,array);
```
- `GetStatus()`
  - Call the `GetStatus` method.  

```
String status = request.GetStatus(token, "ca8af7a0-583a-4a07-9e82a044ba2");
```
- `Get Services()`
  - Call the `GetServices` method.  

```
String[] services = requestor.GetServices(token, "Interfaces");
```

## 2) Document Management API:

Document Management API will be used for downloading, deleting, adding and modifying documents.

- Instantiate the `Node.Core.API.DocManager` class  

```
Node.Core.API.DocManager dm = new Node.Core.API.DocManager();
```
- Uploading a document
  - Instantiate `Node.Core.Document.NodeDocument` and define the name, type and content using the doc.properties name, type, content, and/or Stream properties.
  - Upload the document using `docManager.UploadDocuments()` method
- Downloading a document
  - Call the `docManager.GetDocuments()` method
- Removing a file
  - Call the `docManager.RemoveDocuments()` method

## 3) Node Logging API:

Node Logging API will be used for managing the logs- creating, updating, and removing logs. This API is connected to the database server and makes it easier to manage the database objects that store logs.

An example below shows the implementation of this API for creating and updating log status.

- Instantiate the `Node.Core.API.Logging` class  

```
Node.Core.API.Logging logger = new Node.Core.API.Logging();
```
- Updating Log status
  - Call `UpdateOperationLog` method

## 3.2 Java Version:

The following APIs are available for the Node:

- 1) Document Management API
- 2) Node Requestor API
- 3) API to update the current status of a request (i.e. Node Logging API)

Each API could be called by following two steps:

- 3) Call the constructor for the API
  - 4) Call the methods defined in the API.
-

Some of the Node API's require a connection to the Node Database. All database connection string can be found from Web.xml which is located in the Web-INF directory of the application. One kind of connection way is looks like below:

```
<env-entry>
    <env-entry-name>node/dbFullString</env-entry-name>
    <env-entry-value>

        (description=(address_list=(address=(host=kent) (protocol=tcp) (port=1521)) (add
ress=(host=kent) (protocol=tcp) (port=1521))) (source_route=yes) (connect_data=(sid=ken
tdb)))

    </env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
<env-entry>
    <env-entry-name>node/dbType</env-entry-name>
    <env-entry-value>oracle10g</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
<env-entry>
    <env-entry-name>node/dbUser</env-entry-name>
    <env-entry-value>enode</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
<env-entry>
    <env-entry-name>node/dbPassword</env-entry-name>
    <env-entry-value>enode</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Please check database connection more details in EN-Node Installation Guide.

### 1) Node Requestor API:

Node Requestor API utilizes a method for each Web Service.

- This API utilizes a constructor , `Node.WebServices.Requestor.NodeRequestor()` for version1 or `Node2.webservice.Requestor.NodeRequestor()` for version2.
  - Instantiate the constructor. This object specifies the URL of the node to which the request is made and a logger of application. The logger can be "Phrase.[AdministrationLoggerName](#)", "Phrase.[TaskLoggerName](#)" and "Phrase.[WebServicesLoggerName](#)". If the application want to write log file to Task log file, we should fill Phrase.[TaskLoggerName](#) in second parameter. If the application want to write log file to WebService log file, we should fill Phrase.[WebServicesLoggerName](#) in second parameter.
  - - `NodeRequestor requestor = new NodeRequestor("https://test.epacdxnode.net/cdx/services/NetworkNodePortType_V10", Phrase.TaskLoggerName);`
  - `NodePing()`
    - Instantiate the built-in class String. Call the method `NodePing()` on the constructor with the initial argument as "hello"  
`String ping = requestor.NodePing("hello");`
  - `Authenticate()`
    - Instantiate the built-in class String. Call the method `Authenticate()` on the constructor with the initial arguments as ("cdx", "test", "PASSWORD")
-

- ```
String token = requestor.Authenticate("cdx","test","PASSWORD");
```
- Submit()
    - Instantiate the class Node.Core.Document.NodeDocument  
`NodeDocument doc = new NodeDocument();`
    - Set the name, type, and content or Stream properties of an instantiated Node.Core.Document.NodeDocument class.  
`NodeDocument doc = new NodeDocument();`  
`doc.name = "Test.xml";`  
`doc.type = "XML";`  
`doc.Stream = new MemoryStream();`
    - Call the Submit method.  
`string transID = requestor.submit(token,null,null,array);`
  - GetStatus()
    - Call the GetStatus method.  
`String status = request.GetStatus(token, "ca8af7a0-583a-4a07-9e82a044ba2");`
  - Get Services()
    - Call the GetServices method.  
`String[] services = requestor.GetServices(token, "Interfaces");`

## 2) Document Management API:

Document Management API will be used for downloading, deleting, adding and modifying documents.

- Instantiate the Node.API.DocumentManagement class  
`Node.API.DocumentManagement dm = new`  
`Node.API.DocumentManagement(Phrase.WebServicesLoggerName);`
- Uploading a document
  - Instantiate Node.API.DocumentManagement and define the name, type and content using the doc.properties name, type, content, and/or Stream properties.
  - Upload the document using docManager.Upload () method
- Downloading a document
  - Call the docManager.Download(transactionID) method
- Removing files
  - Call the docManager.Remove (transID,NameList) method

## 3) Node Logging API:

Node Logging API will be used for managing the logs- creating, updating, and removing logs. This API is connected to the database server and makes it easier to manage the database objects that store logs.

An example below shows the implementation of this API for creating and updating log status.

- Instantiate the Node.API.NodeUtils class  
`Node.API.NodeUtils utils = new Node.API.NodeUtils ();`
- Updating Log status
  - Call UpdateOperationLog method

More detail of API please refers Java Doc.

---

## 4 Node Database Details

### 4.1 Configuring Database Connection Settings

#### 4.1.1 .Net Version:

The database connection strings are defined within the Web.config file present in all four components of the Node (Node.TaskHandler.exe.config for the Task Scheduler application). The Web.config file is different for each component and so for a change in the database connection, care should be taken to update only the corresponding Web.config file. As shown below, the database connection string as defined in the connectionStrings block.

```
<connectionStrings>
  <add name="node" connectionString="Data Source=ATHENDB.ENFOTECH.COM;
    User ID=ASPNODE;Password=ASPNODE" providerName="System.Data.OracleClient"/>
</connectionStrings>
```

The name should be node, the connectionString should be in the format of a standard SQL Server or Oracle connection string as referenced below, and the providerName should be System.Data.SqlClient for a SQL Server database or System.Data.OracleClient for an Oracle database.

**SQL Server connection string:** Server=server;Database=database;UID=user;PWD=password

**Oracle connection string:** Data Source=tnsname;User ID=user;Password=password

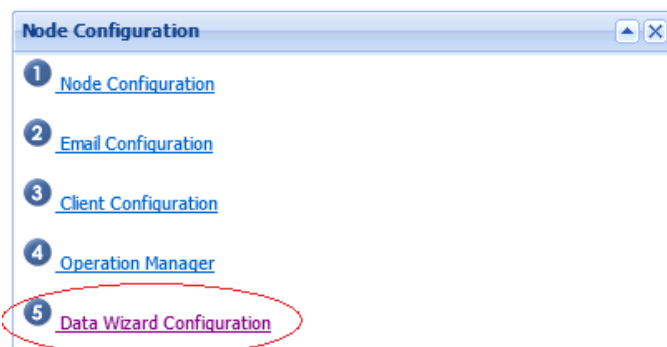
#### 4.1.2 Java Version:

Please refer EN-Node Installation Guide

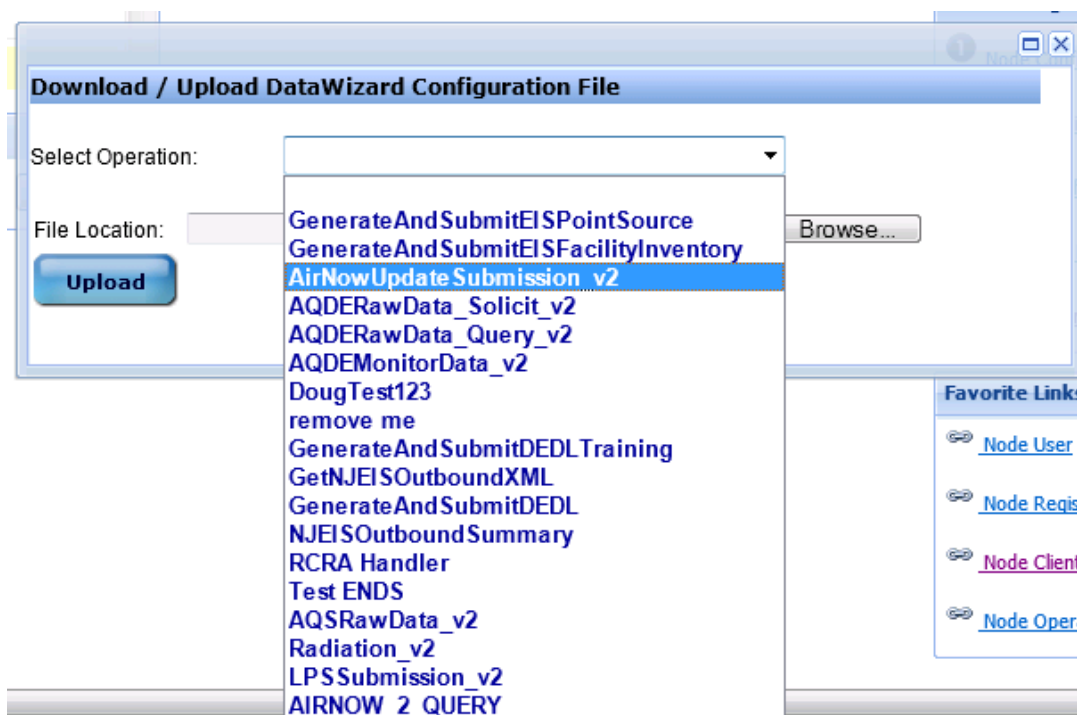
## 5 Manually Update Node Wizard Configuration File (Java Version)

### 5.1 Download wizard BEPL configuration file

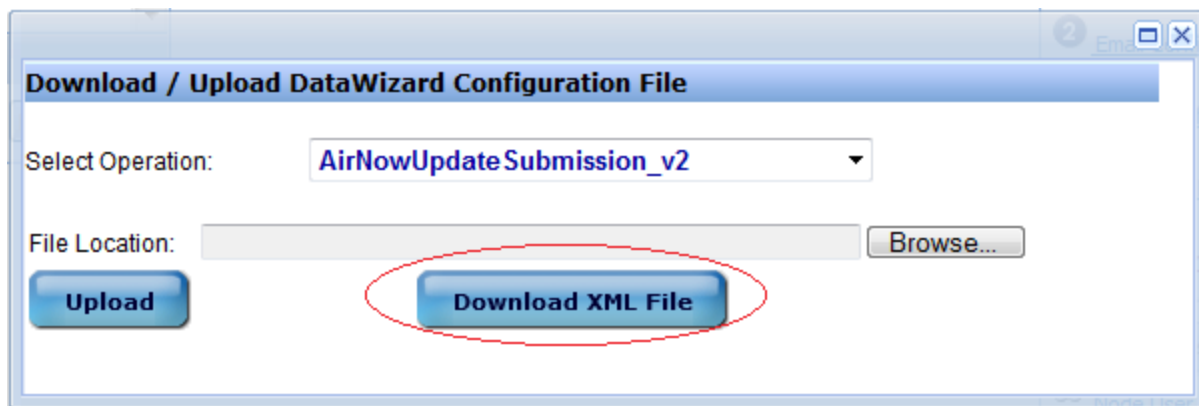
1. Go to the Node console and find the “Data Wizard Configuration”



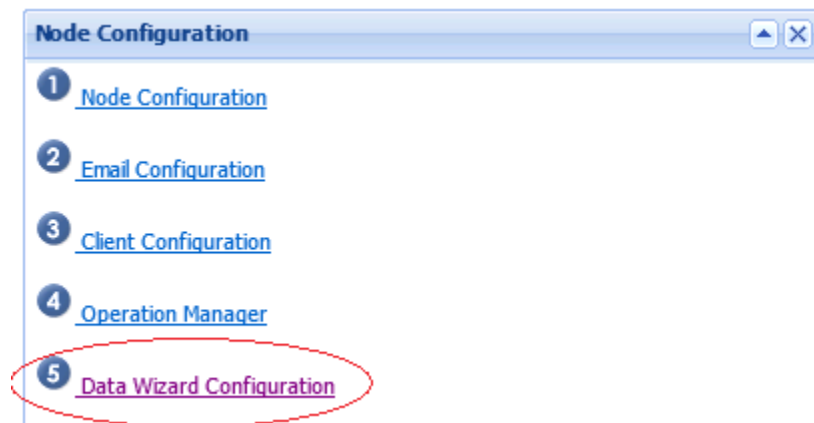
2. Find the operation from the dropdown list



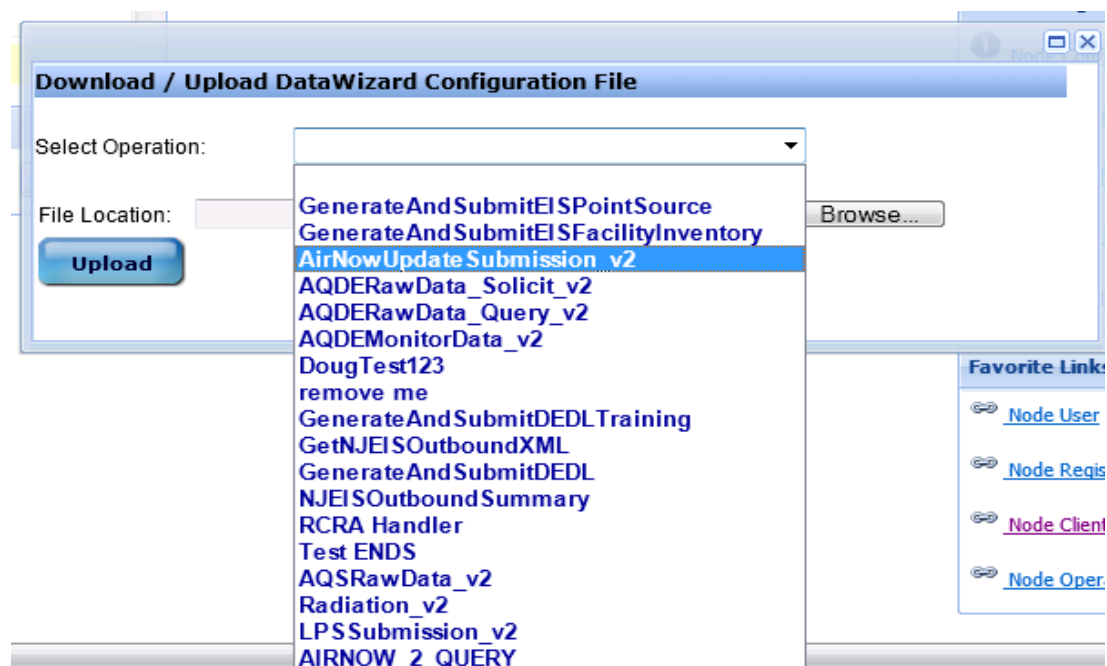
3. Click “Download XML File” button to download the file.



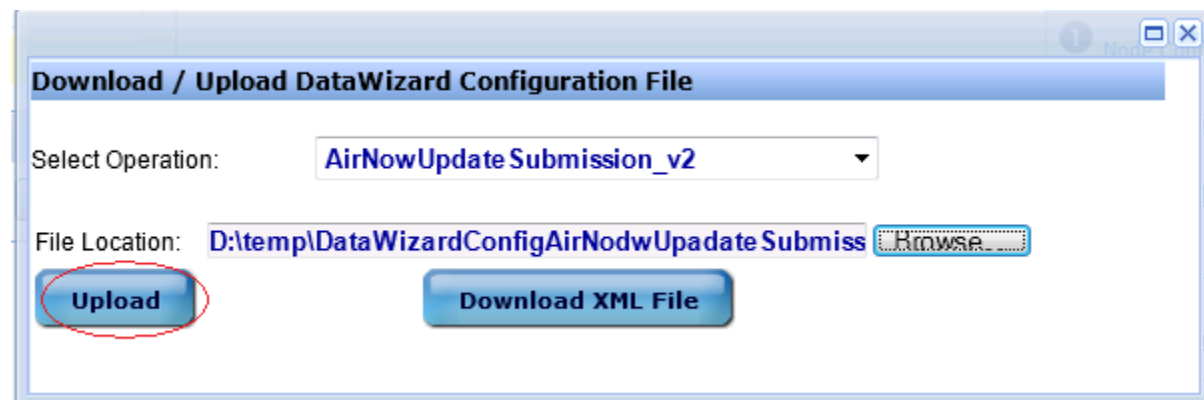
4. After you finish update the wizard BEPL file, please go to the Node console and find the “Data Wizard Configuration”



5. Find the operation from the dropdown list



6. Find the configuration file and click "Upload" button to upload.





## 6 Data Structure

### 6.1 Data Dictionary

Table	Column	Data Type	Description
NODE_ACCOUNT_TYPE			This table stores the various possible account types that could be associated to the user.
	ACCOUNT_TYPE_ID	NUMBER(10)	Primary key for the table
	ACCOUNT_TYPE	VARCHAR2(20)	Account type: Node Admin, Domain Admin, NAAS managed user, locally managed user
	ACCOUNT_DESC	VARCHAR2(50)	Details about the account type
NODE_ACCOUNT_TYPE_XREF			This table cross references the user account type with the User ID and Domain ID.
	ACCOUNT_TYPE_XREF_ID	NUMBER(10)	Primary key for the table
	ACCOUNT_TYPE_ID	NUMBER(10)	Foreign key from NODE_ACCOUNT_TYPE table
	USER_ID	NUMBER(10)	Foreign key from SYS_USER_INFO table
	DOMAIN_ID	NUMBER(10)	Foreign key from NODE_DOMAIN table
NODE_DOMAIN			Stores details about the domain.
	DOMAIN_ID	NUMBER(10)	Unique Domain ID
	DOMAIN_NAME	VARCHAR2(50)	Domain name
	DOMAIN_DESC	VARCHAR2(100)	Description of the domain
	DOMAIN_STATUS_CD	VARCHAR2(10)	Indicates whether the Domain is Running or Stopped
	DOMAIN_STATUS_MSG	VARCHAR2(1000)	Provides a message to Node users regarding the current status of the Domain
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2 (50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2 (50)	Record updated by person's login name
NODE_DOMAIN_WEB_SERVICE_XREF			Acts as a XREF table between the Domain table and Web Service table. This table links the Web Services that are available to the Domain.
	DOMAIN_ID	NUMBER(10)	Foreign key from Domain table ( Part of Composite Primary key)
	WEB_SERVICE_ID	NUMBER(10)	Foreign key from NODE_WEB_SERVICE table ( Part of Composite Primary key)
NODE_FILE_CABIN			This table stores files that are used in Node operations like the Submit and Solicit methods.
	FILE_ID	NUMBER(10)	Unique file ID (Primary key for the table)
	TRANS_ID	VARCHAR2(50)	Node transaction ID which uses files

Table	Column	Data Type	Description
	FILE_NAME	VARCHAR2(200)	File name if it's available
	FILE_TYPE	VARCHAR2(50)	File type if it's available
	STATUS_CD	VARCHAR2(10)	This is for the Solicit method when the URL address is passed to Node. Possible values are R (ready to submit) and S (Submitted)
	DATAFLOW_NAME	VARCHAR2(50)	Indicates the data flow either supplied as a parameter during a Submit, or supplied by the Node administrator during file upload.
	SUBMIT_URL	VARCHAR2(100)	The URL to which documents will be submitted. This is for the Solicit method when the URL address is passed to Node.
	SUBMIT_TOKEN	VARCHAR2(200)	The security token that the application will use to invoke the Submit method. This is for the Solicit method when the URL address is passed to Node.
	SUBMIT_DTTM	DATE	Date time that the document is submitted to URL. This is for the Solicit method when the URL address is passed to Node.
	FILE_CONTENT	BLOB	The content of file in BLOB format.
	FILE_SIZE	NUMBER(10)	Stores the size of the document
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2(50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2 (50)	Record updated by person's login name
<b>NODE_OPERATION</b>			<b>Stores information related to the operations</b>
	OPERATION_ID	NUMBER(10)	Unique operation ID auto created by the system.
	DOMAIN_ID	NUMBER(10)	Foreign key from Domain table that identifies the unique domain for which this operation is associated.
	WEB_SERVICE_ID	NUMBER(10)	Foreign key from Web_Service table that identifies the unique web service handler for which this operation is associated.
	OPERATION_NAME	VARCHAR2(50)	Name of the operation
	OPERATION_DESC	VARCHAR2(255)	Description of the operation
	OPERATION_TYPE	VARCHAR2(25)	Indicates the operation type as either "Scheduled Task" or "Web Service"
	OPERATION_CONFIG	XMLTYPE	Stores details about operations
	OPERATION_STATUS_CD	VARCHAR2(20)	Indicates the status of the operation.
	OPERATION_STATUS_MSG	VARCHAR2(1000)	Stores the status message provided by the Domain Administrator for the operation.
	VERSION_NO	VARCHAR2(10)	Stores the version of Node (VER_11 or VER_20)
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2(50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2(50)	Record updated by person's login name
<b>NODE_OPERATION_LOG</b>			<b>Logs in the details of the web service operations.</b>
	OPERATION_LOG_ID	NUMBER(10)	Primary key for the table auto created by the system

Table	Column	Data Type	Description
	TRANS_ID	VARCHAR2(50)	Unique transaction ID generated for each service request
	OPERATION_ID	NUMBER(10)	Foreign key from Operation table that identifies the unique operation for which this operation log applies (if the system could resolve one based on the service request).
	USER_NAME	VARCHAR2(50)	Identifies the USER_NAME either directly supplied (for Authentication requests), or obtained via security token and previous operation log.
	REQUESTOR_IP	VARCHAR2(30)	IP address of the Node requestor.
	SUPPLIED_TRANS_ID	VARCHAR2(50)	Transaction ID supplied as an input parameter for Submit, GetStatus, and Download service requests.
	TOKEN	VARCHAR2(200)	Security token that is used to make Node request.
	NODE_ADDRESS	VARCHAR2(100)	Nodeaddress parameter supplied as part of Notify service request
	RETURN_URL	VARCHAR2(100)	returnURL parameter supplied as part of Solicit service request
	SERVICE_TYPE	VARCHAR2(50)	ServiceType parameter supplied as part of GetServices service request
	START_DTTM	DATE	Date time a task started (for scheduled operations) or the datetime the service request was made.
	END_DTTM	DATE	Date time a task completed (for scheduled operations) or the datetime the service request response was made.
	HOST_NAME	VARCHAR2(40)	Server name which ran the task (for scheduled operations).
	CREATED_DTTM	DATE	Record created by person's login name
	CREATED_BY	VARCHAR2(20)	Record created date time
	UPDATED_DTTM	DATE	Record updated by person's login name
	UPDATED_BY	VARCHAR2(20)	Record updated date time
NODE_OPERATION_LOG_PARAMETER			Stores each parameter supplied as part of the parameter array for Query and Solicit service requests. Allows for relating asynchronous requests having different tokens.
	OPERATION_LOG_ID	NUMBER(10)	Unique operation log that helps form composite primary key for the table
	PARAMETER_NAME	VARCHAR2(50)	Parameter name supplied as part of service request that helps form composite primary key for the table.
	PARAMETER_VALUE	VARCHAR2(200)	Stores the value for the supplied parameter
NODE_OPERATION_LOG_STATUS			Stores details about the status of the operation logs
	OPERATION_LOG_STATUS_ID	NUMBER(10)	Primary key for the table
	OPERATION_LOG_ID	NUMBER(10)	Foreign key from Node_Operation_Log table
	STATUS_CD	VARCHAR2(20)	Status of the operation
	MESSAGE	VARCHAR2(4000)	Message for the operation
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2(50)	Record created by person's login name
NODE_USER_OPERATION			Table to manage privileges of the user with respect to the operation

Table	Column	Data Type	Description
<b>_XREF</b>			
	USER_ID	NUMBER(10)	Foreign key from SYS_USER_INFO table (Part of Composite Primary Key)
	OPERATION_ID	NUMBER(10)	Foreign key from NODE_OPERATION table (Part of Composite Primary Key)
<b>NODE_WEB_SERVICE</b>			Stores details about the available Web Services
	WEB_SERVICE_ID	NUMBER(10)	Primary key for the table
	WEB_SERVICE_NAME	VARCHAR2(50)	Name of the Web Service
	WEB_SERVICE_DESC	VARCHAR2(50)	Description of the Web Service
<b>SYS_ADDRESS</b>			Stores info about address for the user accounts
	ADDRESS_ID	NUMBER(10)	Primary key for the table
	ADDRESS	VARCHAR2(100)	User address
	SUPPLEMENTAL_ADDRESS	VARCHAR2(100)	Supplemental address
	LOCALITY_NAME	VARCHAR2(100)	Address locality name
	STATE_CD	CHAR(2)	Address state code
	ZIP_CODE	VARCHAR2(15)	Address zip code
	COUNTRY_CD	VARCHAR2(25)	Address country code
	STATUS_CD	CHAR(1)	Active/Inactive status
	ADDRESS_DESC	VARCHAR2(100)	Description of the address
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2(50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2(50)	Record updated by person's login name
<b>SYS_CONFIG</b>			This table stores node system configuration files. Those files are used by application and will effect how the system runs.
	CONFIG_ID	NUMBER(10)	Unique configuration ID
	CONFIG_NAME	VARCHAR2(300)	Configuration file name
	CONFIG_XML	XMLTYPE	Content of configuration file
	CONFIG_TYPE_CD	VARCHAR2(25)	Indicator to differentiate the type of configuration file
	STATUS_CD	CHAR(1)	Configuration file status. Possible values are A (Active) or I (Inactive). This must be set to active "A" for the system to read this file.
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2 (50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2 (50)	Record updated by person's login name
<b>SYS_EMAIL</b>			Stores info about email for the user accounts
	EMAIL_ID	NUMBER(10)	Unique ID for the table
	EMAIL_ADDRESS	VARCHAR2(100)	Email address (ID) for the user account

Table	Column	Data Type	Description
	EMAIL_TYPE	VARCHAR2(25)	For differentiating Emails when User has more than one Email ID
	STATUS_CD	CHAR(1)	Active/Inactive status
	CREATED_DTTM	DATE	Record created by person's login name
	CREATED_BY	VARCHAR2(20)	Record created date time
	UPDATED_DTTM	DATE	Record updated by person's login name
	UPDATED_BY	VARCHAR2(20)	Record updated date time
SYS_SEQUENCE_NO			Table used to manage automatic numbering of the primary keys for other tables.
	SEQUENCE_ID	NUMBER(10)	Primary key for the table
	TABLE_NAME	VARCHAR2(30)	Table for which the sequence number is created
	COLUMN_NAME	VARCHAR2(30)	Column for which the sequence number is created
	MIN_NUMBER	NUMBER(5)	Minimum number for the sequence
	MAX_NUMBER	NUMBER(10)	Max number limit for the sequence
	LAST_USED_NUMBER	NUMBER(10)	Displays the last used sequence number for the column in that specific table
	STATUS_CD	CHAR(1)	Active/Inactive status
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2(50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2(50)	Record updated by person's login name
SYS_USER_ADDRESS			Maps user to their Address via SYS_ADDRESS table
	ADDRESS_ID	NUMBER(10)	Unique Address ID (Part of Composite Primary Key)
	USER_ID	NUMBER(10)	Unique User ID from SYS_USER_INFO table (Part of Composite Primary Key)
	ADDRESS_TYPE_CD	VARCHAR2(25)	Specifies the type of address
SYS_USER_EMAIL			Maps users to their Email ID via SYS_EMAIL table
	EMAIL_ID	NUMBER(10)	Unique Email ID (Part of Composite Primary Key)
	USER_ID	NUMBER(10)	Unique User ID from SYS_USER_INFO table (Part of Composite Primary Key)
SYS_USER_INFO		-	This table stores user account information for using Node administration tool. System will use information from this table to control user login Node administration and security rights of using Node administration tool.
	USER_ID	NUMBER(10)	Unique user ID assigned
	LAST_NAME	VARCHAR2(60)	Node user's last name
	FIRST_NAME	VARCHAR2(60)	Node user's first name
	MIDDLE_INITIAL	CHAR(1)	Node user's middle initial
	LOGIN_NAME	VARCHAR2(50)	Login name for the user
	LOGIN_PASSWORD	VARCHAR2(100)	Login password for administration user account
	USER_STATUS_CD	VARCHAR2(10)	User account status code. This controls if the user can login to the system. Possible values are A (Active) or I (Inactive).

Table	Column	Data Type	Description
	LAST_4_SSN	VARCHAR2(4)	Last 4 digits of admin's social security number
	CHANGE_PWD_FLAG	VARCHAR2(1)	A flag which indicates if password need to be changed when user login in. Possible values are Y or N. Flag will be set to Y when user's password has been created for 1st time. After the password has been reset when the user logs in, flag will be set to N.
	PHONE_NUMBER	VARCHAR2(15)	User contact phone number
	COMMENTS	VARCHAR2(255)	Comment on user account
	CREATED_DTTM	DATE	Record created date time
	CREATED_BY	VARCHAR2(50)	Record created by person's login name
	UPDATED_DTTM	DATE	Record updated date time
	UPDATED_BY	VARCHAR2(50)	Record updated by person's login name
	UPDATED_DTTM	DATE	Record updated date time
NODE_PROCESS_MONITOR		-	This table stores background process information.
	PID	INTEGER	Primary key for the table
	TRANSACTION_ID	VARCHAR(100)	Unique transaction ID generated for each service request
	OPERATION_ID	INTEGER,	Foreign key from Operation table that identifies the unique operation for which this operation log applies (if the system could resolve one based on the service request).
	PROCESS_NAME	VARCHAR(300)	Name of Process under threading condition
	PROCESS_STATUS	VARCHAR(100)	Status of Process under threading condition
	UPDATED_DTTM	DATETIME,	Record updated date time
	UPDATED_BY	VARCHAR(100),	Record updated by person's login name
	NODE_ADDRESS	VARCHAR(300)	Record the external URL which is calling by EN-Node

## 6.2 Database Diagram

