DEMONSTRATED NODE CONFIGURATION FOR THE STATE OF MAINE

Last Updated: February 12, 2004

Version: Version 1.1 Draft

Authors: Calvin Li, Oracle Corporation
Robert E. Williams , Maine Bureau of Information Services
David H. Ellis, Maine Bureau of Information Services

Contents

Introduction

Introduction

This guide provides the detail documentation on the EPA Exchange Network (EN) Node implementation of the State of Maine produced by using Oracle 9iAS and an Oracle 9i database.

Audience

This document is targeted at the administrator and developer levels.

The administrator is someone who sets up and administrates application and database servers. The administrator is responsible for installing, configuring and managing the 9i database and 9iAS application server.

The developer is the person who develops the Node Web Services application. The developer is responsible for coding Java, PL/SQL and producing EAR files and PL/SQL packages for an administrator to deploy.

Chosen Approach

The objectives of implementing the Maine Node are to:

1. Comply with the Network Node Functional Specification v1.1.
2. Comply with the standards of Web Services (WSDL, UDDI and SOAP).
3. Use Oracle 9iAS and 9iDB technology as the infrastructure for the Maine Node.
4. Enhance and develop new modules easily in future.
5. Provide interoperability with all Nodes regardless of how they are built.

In order to achieve those objectives, we chose the following approach for the Maine Node development:

a. Use Java/J2EE to develop the Web services. It can be developed and deployed on any platform. J2EE is the default framework to develop Web Services.
b. Use Oracle 9iAS as the platform for the Node Server, which supports the J2EE Application and SSL the Maine Node Security.
c. Use JDeveloper to develop the application and generate the EAR files. JDeveloper can be installed on any platform (Maine used JDeveloper on Windows 2000).
d. JDeveloper is not used to deploy the application on 9iAS because of a security issue. With JDeveloper 9.0.3 the connection to the application server required the use of the Admin account. During the testing of the connection, the password for the account was displayed, unencrypted, in the JDeveloper IDE. Instead, JDeveloper generates EAR files, and the administrator is responsible for deploying the EAR files.
e. Use Oracle 9iDatabase for meta-data tables, data storage, service tracking, and XML processing
    i. The entire XML document for QUERY method is created using the XML toolkit (XSU) in the 9i (9.0.2) database.
    ii. XML toolkit is used to create an XML document from the SQL Query, and then a Style Sheet is applied to deliver the XML document in the format required by the XML schema in the Functional Specification v1.1.
f. Multi-Tier Architecture.
g. Use Apache-Axis which implements the JAX-RPC specification standard and supports the DIME attachment protocol for larger files.

Software

Software Dependency

The following software must be installed before beginning the installation of the Node.

Software	Version
Oracle 9i Database Server	9.2.0.3
Oracle 9iAS Release 2	9.0.3
SSL Certificate	128-bit encryption, Application Server-specific

Other Dependencies

Each Node must have an account set up in order to differentiate its transactions from those of other Nodes and to identify it for EN security purposes. To create a Node administrator account for the Exchange Network, contact the CDX Help Desk at Nodehelpdesk@csc.com or 1-888-890-1995. **Do not use this administrator account for testing the node. Create a separate operator account through the CDX Help Desk for testing.**

Development Software

Software	Version
Oracle 9i JDeveloper	9.0.3
SQL*Plus or TOAD	
FTP software	
UNIX shell (use for Cron)	
Java Decompiler (optional)	Decafe Pro 3.9

Patches

For additional information regarding SSL certificates and the Oracle Wallet, please refer to Oracle document (Doc ID: Note: 153653.1) in the Application Server notes section, at the end of the DNC.

Hardware

Database Server:

Platform: Sun Sparc Solaris
Operating System Version: Solaris 8

Application Server:

Platform: Sun Sparc Solaris
Operating System Version: Solaris 8

Developer's Computer:

Operating System Version: Windows 2000

The Node Server (Web services) is written in Java. It should support the following platforms.

i. Windows
ii. Linux
iii. Solaris

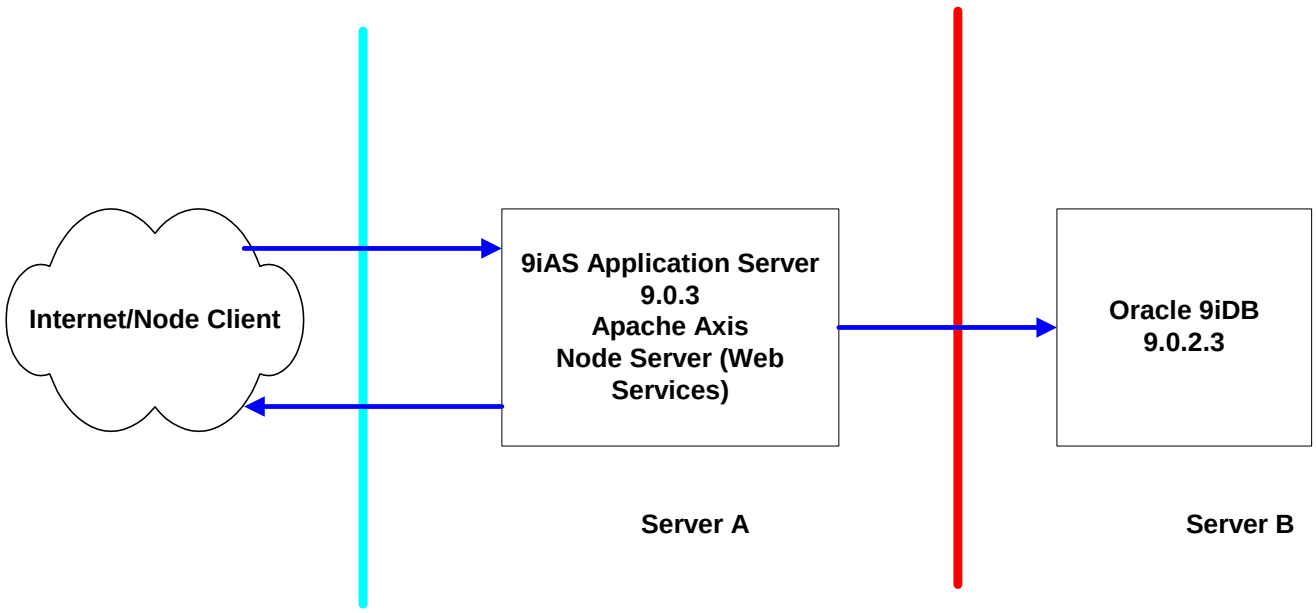
Maine has only used the Solaris platform for the application server and setting up the Node server.

The Deployment Model

For building and deployment of the Node, Maine chose the following model. The application server was set up on one machine, the database on a second machine, and the development IDE (JDeveloper) on a third machine.

The developer would build the Node and deploy it as an .EAR (Enterprise Archive) file. The .EAR file would be sent to the administrator of the application server for deployment. The administrator, using Oracle Enterprise Manager Web interface, would deploy the file.

An additional caveat here regarding the development, versus production, environment: if the Node is behind a firewall, the network administrator will need to create a "tunnel" so that only friendly incoming requests can reach the Node, be processed, and sent back to the requestor.



Installation

For the Installation process, the steps have been rolled up into sections. There should be a separate section addressing the Application Server, Database, and the Java source code, among others. Each section begins with the base requirements and then moves into specific modifications or steps that need to be performed in order to deploy the Node. The hope here was that keeping the information for each element in its own section would prevent unnecessary jumping back and forth between components.

***Note**: As mentioned above in the Software Dependency section, a specific base level of components must be installed prior to implementing the steps below.*

The DNC zip file contains the source code necessary to build the Node. In addition to the Java source code, components needed in the application server and database are included as follows:

DNC\Source_Code\Application_Server – contains components needed in the application server.

DNC\Source_Code\database – contains the database components (.dmp file, etc).

DNC\Source_Code\Node – contains the Java source code for the Node and the SubmitListener (client).

Install and Setup Oracle Application Server (9iAS J2EE)

***Note**: Make sure all other application server installs or infrastructure installs are running. This assures that the new install will allocate different ports if they have been previously allocated.*

It is assumed at this point that Oracle Infrastructure 9.0.2.2.0 has been installed on a Sun Solaris Server This is done by installing the 9.0.2.1.0 core infrastructure and then applying the following patches (in order):

1. OID 9.0.2.2.0 Patch set (Patch # 2559205)
2. 9.0.2.2.0 Core Patch set (Patch # 2703110)
3. 9.0.2.2.0 NC Patch set (Patch # 2926973)

When all of the above installs are completed, you will have installed Oracle 9iAS Release 2 version 9.0.3

Configure the J2EE container (OC4J)

1. Be sure Oracle Enterprise Manager (OEM) is running.
2. Login to the OEM http://<server-name>:<port> using the userId and the password set when installing 9iAS 9.0.3.
   Note: It should be the default port, which is 1810.
3. If you have only one instance installed, you will automatically skip this step: on the initial page after login, choose the 9.0.3 server that you just installed by clicking on the iAS 903 installation link.
4. The Application Server: System Components will appear.
5. On that page, under System Components, click on the Create OC4J Instance button on the middle right-hand side of the screen.
6. Enter OC4J_NODE as the instance name and click Create.
7. The Confirmation page comes up. Click on OK.
8. The Application Server: System Components page is redisplayed.
9. Under System Components, click on the OC4J instance that you just created (OC4J_NODE).
10. On the OC4J_NODE page, click the "Start" button.
11. Click the "OK" button on the confirmation page.

12. On the OC4J_NODE instance page, under Administration, select the Server Properties link. This will take you to the Server Properties page.
13. Under Command Line Options you will see a text box called Java Options. Copy and paste the following lines into that text box (starting with the hyphen and being careful not to include the return character after "…ParserFactoryImpl"). Type in the address manually if, after clicking on the Apply button, you do not receive Confirmation.

    -Djavax.XML.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
    -Djavax.XML.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
14. Click on the Apply button.
15. There will be Confirmation that server properties were applied. Click ok. This will return you to the Server Properties page.
16. You will now need to restart the OC4J instance for the changes to take effect. Select the OC4J_Node link at the top of Server Properties configuration page.
17. Once on the OC4J_NODE page, select the "Stop" button. Select "Yes" to confirm the stop. Once the OC4J instance has stopped, then select the "Start" button.
18. On your PC, in the DNC folder, FTP the following file: ..\Source_Code\Node\Node\conf\properties\user_properties.XML under the **$ORACLE_HOME/j2ee/home** directory on the application server.

    Note: Do not put it under the instance that you created.
19. On your PC, in the DNC folder, FTP the following file: ..\Source_Code\Node\Node\lib\commons\xerces.jar under the **$ORACLE_HOME/jdk/jre/lib/ext** directory on the application server.
20. Again, on the OC4J_NODE page, select the "Stop" button. Select "Yes" to confirm the stop. Once the OC4J instance has stopped, then select the "Start" button.

Set up the XML Style Sheets and Schema

On the application server, create two Web accessible folders (one to hold the XML schema and the other to hold the XML Style Sheets).

To access these folders, modify the httpd.conf file under oraclehome/apache/apache/conf/ by adding an alias for each of two folders using one of the existing aliases as a template. Template example: "alias /icon/".

Inside of the folder: DNC\Source_Code\Application_Server, there should be two folders containing the XML schema and XML Style Sheet files. These files will need to be FTP'd up to the application server.

***Note**: Once the folders are built and files transferred, the path in the PL/SQL stored procedures to these files will need to be changed after all the database work has been done.*

Oracle Wallet Manager

Oracle Documentation: http://download-west.oracle.com/docs/cd/A97329_01/core.902/a90146/owm.htm#1005543

***Note**: Oracle doesn't support the conversion of a 128- bit "Global Server Certificate", but a 128-bit certificate can be used with the Oracle Wallet if you create your certificate request from the wallet. You will need to obtain a SSL certificate for your node's application server. You may obtain one at no charge from EPA by calling the Node Help Desk or you may purchase one from a vendor. In either case, you must follow the steps below to obtain your certificate.*

To create a new Oracle Wallet after the default installation has been completed, please do the following:

1. Open an XWindows emulator (Exceed, X-win32 [a product of StarNet]). Open the Oracle Wallet Manager.
2. Open a terminal session, connecting as the "Oracle" software administrator.
3. Set the environment using the following command: ". orasid ias903". Note: ias903 is set to the 9.0.3 application server home in the oratab file.
4. To determine your PCs IP address, open a new command prompt and type: ipconfig
5. Copy the IP address into following string: DISPLAY=<IP_ADDRESS>; export DISPLAY
6. Paste the string from the above step into the terminal window. Press <Enter>.
7. Type: owm
8. Press <Enter>.
9. On the taskbar, there will be a lock and key minimized window. Click on this and maximize it.
10. On the menu bar select "Wallet → New".
11. A message might appear next stating that your default wallet directory doesn't exist and you will be prompted to create it. Choose "YES".
12. A new informational message might appear stating that it is unable to create your default wallet directory and you will have to save it in an alternate location. You will be asked if you want to continue. Choose "YES".
13. Enter a password for the Oracle Wallet.
14. A new empty wallet has now been created. Choose "YES" when prompted if you want to create a new certificate request.
15. Enter your identity information. Select the 1024 bit key size and press the <Enter> key.
16. Save the newly created Wallet to the directory $ORACLE_BASE/wallet
17. Under "Wallet" in the navigator, right click on "Certificate:[Requested]".
18. Select: "Export Certificate Request".
19. Save the file under $ORACLE_BASE/wallet, calling the file name "cert.req". Press the <Enter> key.
20. Now that the request has been created, it can be used to purchase a certificate from an Authorized Certificate Authority. Please work with the particular CA that you have chosen and purchase a new certificate.
21. After you receive your new certificate, import it into the Oracle Wallet.
22. Under "Wallet" in the navigator, right click on "Certificate:[Requested]".
23. Select: "Import User Certificate".
24. Choose the option to select from a file.
25. After you have imported the certificate and any trusted authorities, you need to export the wallet. Choose "Operations → Export Wallet". Save this as "$ORACLE_BASE/wallet/exp_wallet". SEND this file to the developer to store on their machine for testing purposes.
    Note: $ORACLE_BASE is not the $ORACLE_HOME.
26. Save and close the wallet.
27. To get the Oracle Application Server to recognize the Oracle Wallet, you need to ensure the following lines are setup correctly in the httpd.conf file. Examples are as follows:
    SSLWallet file:$ORACLE_BASE/wallet/
    SSLWallet password <PASSWORD>
    Note: the password can be either clear text or obfuscated. See the Oracle Documentation for more information.
28. Cycle the Oracle Application Server for the wallet changes to take effect.
29. Validate the SSL is in place by opening a Web browser and navigate to a similar URL as the following: https://<SERVERNAME>:<SSL PORT>
30. Double-click on the padlock in the bottom right side of the browser. A certificate window should appear with the following credentials:
    Issued to: <SERVERNAME>
    Issued By: <An authorized Certificate Authority>
    Valid From: <DATE> to <Date>

Setting up the Job Scheduler (CRON)

Under the folder: DNC\Source_Code\Node\Application_Server, there is a directory structure to implement a job scheduler in CRON (UNIX). If you are not using the UNIX job scheduler, the files will need to be modified for your platform.

In the DNC, navigate down to the folder: ..\DNC\Source_Code\Application_Server\cron. In this folder are scripts to set paths, execute PL/SQL scripts, and execute a .jar file that will call the submit method on a Node that has called the Solicit method on your Node. These files will have to be adjusted to your environment.

Directories:

- jar: submitListener.jar for running the submitListener
  ***Note**: Place the created .jar file here (see Prepare Deployment Files).*
- other: contains path information for the scripts
- script: UNIX script that runs the SQL scripts and Java file
- sql: contains SQL scripts that run database procedures

Files:

- cron_job: UNIX automatic job execution script
- set_path: sets path for the UNIX environment

Once you have changed the scripts for your environment, perform the following:

1. FTP the Cron folder and its subfolders up to the application server.
2. Login with a UNIX shell.
3. Copy the script **set_path** into the session.
4. Type: crontab –e
5. Copy the script **cron_job** into a crontab –e session.
6. Exit the crontab session.

***Note**: Until ready to test or the other sections have been completed, comment out the Cron job.*

Install And Setup The Oracle Database

- Install the Oracle 9i Database Software and create a core database.
  Note : Refer to the Oracle 9i Installation Guide.
- Start the database and listener. When you are able to connect to the database, you are ready to start executing the custom Node scripts.
- Log into SQL*Plus as "sys" user. Load the XML component into the Database from a telnet session run the following in SQL*Plus:

```
SELECT object_name
     , object_type
     , status
  FROM dba_objects
 WHERE status = 'INVALID';
```

***Note**: There should be no rows returned. If there are rows returned, there may have been problems with the installation. Try to recompile the objects that are invalid. Refer to the Oracle documentation for assistance.*

```
@$ORACLE_HOME/rdbms/initXML.sql

SELECT object_name
     , object_type
     , status
  FROM dba_objects
 WHERE status != 'INVALID';
```

***Note**: There should be no rows returned. If there are rows returned, there may have been problems with the execution of the script. Try to recompile the objects that are invalid. Refer to the Oracle documentation for assistance.*

Creation of Tablespaces

***Note**: The Node stores transaction data, transaction logs, and failed messages. Optionally, payloads are stored in the Oracle Database.*

```
CREATE TABLESPACE depNode01ts
  DATAFILE
    '/<drive>/oradata/<schema>/depNode01ts01.dbf' SIZE 50m
      AUTOEXTEND OFF
  PERMANENT ONLINE
  EXTENT MANAGEMENT LOCAL
      UNIFORM SIZE 100k
;

CREATE TABLESPACE depNode01xs
  DATAFILE
    '/<drive>/oradata/<schema>/depNode01xs01.dbf' SIZE 25m
      AUTOEXTEND OFF
  PERMANENT ONLINE
  EXTENT MANAGEMENT LOCAL
      UNIFORM SIZE 100k
;

CREATE TABLESPACE depNode01_lob
  DATAFILE
    '/<drive>/oradata/<schema>/depNode01_lob01.dbf' SIZE 50m
      AUTOEXTEND OFF
  PERMANENT ONLINE
  EXTENT MANAGEMENT LOCAL
      UNIFORM SIZE 500k
;
```

Creation of User

```
CREATE USER DEPNODE IDENTIFIED BY depNode
DEFAULT TABLESPACE DEPNODE01TS
TEMPORARY TABLESPACE TEMP
PROFILE DEFAULT
QUOTA UNLIMITED ON depNode01ts
QUOTA UNLIMITED ON depNode01xs
QUOTA UNLIMITED ON depNode01_lob
QUOTA UNLIMITED ON users
;
```

- Role Grants

```
GRANT AQ_ADMINISTRATOR_ROLE TO DEPNODE;
GRANT CONNECT TO DEPNODE;
GRANT RESOURCE TO DEPNODE;
GRANT EXECUTE ON DBMS_AQ TO DEPNODE;
GRANT EXECUTE ON DBMS_AQADM TO DEPNODE;
REVOKE UNLIMITED TABLESPACE FROM DEPNODE;
```

Setup the Oracle Database

Grant Java permissions. The database generates the XML document using the XSU package. The database needs to access the Style Sheet in order to generate the XML document, so the database administrator needs to grant permission to the schema for accessing the Style Sheets. Run the following PL/SQL statements (<schema_name> and <host_name> should be in uppercase):

```
exec dbms_java.grant_permission('<schema_name>',
'SYS:java.net.SocketPermission','<host_name>','resolve');
```

***Note**: host_name is the name of the server which has the style sheets. For example: <server_name>.dep.state.<state_abbreviation>.us*

```
exec dbms_java.grant_permission('<schema_name>',
'SYS:java.net.SocketPermission', '<host_name>:<port>', 'connect, resolve');
```

***Note**: IP address and port = IP address of the host name and the port number of the application server. For example: <server_name>.dep.state.<state_abbreviation>.us:80*

You are now ready to import the data using the depnodedba.dmp file provided. After setting the $ORACLE_SID, $ORACLE_HOME, using the following ". orasid <*ORACLE_SID*>" (this is tied to the oratab file) for the session, run the following command:

```
imp system file= depnodedba.dmp log=exp_XMLNode.log buffer=1048576
fromuser=depNode touser=depNode ignore=y rows=y commit=y
```

Once the database is in place and everything is compiled, execute the following statement to start up the QUE. Use SQL*Plus to login to the created schema, then type:

```
exec dbms_aqadm.start_queue('SOLICIT_JOB_QUEUE');

exec dbms_aqadm.start_queue('SUBMIT_JOB_QUEUE');
```

In the folder: ..\DNC\Source_Code\database\PL_SQL there is a SQL script called: developer_scripts. Place the username (CDX helpdesk account for your Node) into the first script and run this script in the SQL*Plus session. This should add your username to the user_profile and user_authorization tables. If you do not do this, you will not be permitted access to the Web services provided by your Node. Close the SQL*Plus session.

Adjust stored procedures to reflect the correct Application Server path of the stylesheets and schemas.

Note: Once the folders are built and files transferred, the path in the PL/SQL stored procedures to these files will need to be changed.

Prepare Deployment files (Java Source code)

The code and build scripts are already configured in the JDeveloper workspace included in the DNC. The scripts should build two files: **Node.ear** and **SubmitClient.jar**

Start up JDeveloper.

Inside JDeveloper, click on File -> Open, open the DNC workspace, navigate to ..\DNC\Source_Code\Node\, click on: **Node.jws**

In JDeveloper, expand the workspace. There should be two projects: Node.jpr and SubmitClient.jpr

Create Node.ear

1. Under the Node.jpr, navigate to the source package gov.maine.Node. Here you will edit the source code Node.java.

2. In Node.java, change the static String variables: “connectString” and “db” to the database server connection information for your environment (get this information from your DBA).

3. Check the createDBConnection method and update the database connection information for your environment. Change the schema userId and password for the DriverManager to reflect your environment (get this information from your DBA).

4. Right Click Node.jpr

5. Select Rebuild Node.jpr

6. Node.ear is created in ../source_code/node/node/dist/

Create SubmitClient.jar

1. You will need to add a library in JDeveloper. Right click on the SubmitClient.jpr project.

2. Select "Project Settings".

3. Click on the “Libraries” folder in the Navigator on the left-hand side of the screen.

4. Under "Selected libraries", click on the file 9.2.JDBC.

5. Click on the "Edit" button.

6. Click on the "Edit" button beside the ClassPath field.

7. In the window that comes up, click on the entry and click "Remove".

8. Next, click on the "Add Entry" button.

9. In the "Select Path Entry" window that comes up, navigate down into the DNC SubmitClient project folder and into the ..\lib directory below it.

10. Click on the Classes12.jar file and Select it.

11. Click “OK” three times to close all the open windows and accept the changes.

12. Under SubmitClient.jpr, edit the file SubmitListener.java.

13. Inside SubmitListener.java, change the information for the following static variables: connectString, userId, password and db (get the information for your database from your DBA).

14. Also in SubmitListener.java, on the line after the comment “authenticate with the node and receive a token”, change <node_user_account>,<user_password> to reflect your CDX Node user account information. Do not change [space]`"password"`.

15. Under NetworkNodeStub.java, Look below the comment section “Oracle SSL Workaround”. Change the Oracle wallet (the exported wallet) path information and wallet password to that for your environment. The path information should include the name of the “exported” wallet you’re your Application Server administrator).

16. Under SubmitClient.jpr, open the folder marked “deployment”. If the folder is not there, look for the deployment file in step 5.

17. Right click on the file SubmitClient.deploy. Choose the option "deploy to Jar".

18. The jar file (SubmitClient.jar) should be deployed to:
    <Node_project_directory>/SubmitClient/deploy/

19. FTP the created .jar file up to the application server folder ../cron/jar/ (created while performing the installation of the application server above).

***Note**: The SubmitClient.jar piece of code is part of what does the asynchronous processing of calls to the Solicit method. It makes the call to the Submit method when the requestor has given a returnURL in the call to the Solicit method.*

Deploying the Node

Node Server Web Services

1. Login to 9iAS 9.0.3 Oracle Enterprise Manager: https://<application_server_name>:1810

2. Click on the application server link for the application server that you installed (or the one you are using) for this application.

3. Click on the link for the OC4J instance that you created above.

4. Make sure the OC4J_NODE status is active (green check).

5. Click the button "deploy EAR" to deploy the EAR.

6. On the Select application page (step two), click on Browse and drill down to the folder containing the Node.ear file that you built. Select the Node.ear and click the Open button.

7. Type "Node" for the application Name.

8. Click Next.

9. The next page will display Web Applications and their URL mappings, which are included in Node application. **Do not make any changes to this page**.

10. Click Finish.

11. The Deploy Application Summary page will come up. No Web Services will be found. The Enterprise Manager is looking for the Oracle Web service toolkit. We are using Apache AXIS instead. Click Deploy.

12. You have deployed the application. To verify, go to https://{serverip:port}/Node/ . A page with "Apache – AXIS" at the top should appear.

Important Notes!!!!

1. Make sure the Style Sheet address in the SQLQUERY2.PLB procedure is correct. Change the Style Sheet address if necessary.

2. Make sure the database connection information in the Node.ear is correct. If not, the developer needs to use JDeveloper to edit the file Node.java (createDBConnection method) in Node.jpr, and rebuild the EAR, which has the updated database connection information.

Testing the Deployed Node

Testing using the included Client:

After verifying above that the Node has been deployed successfully, it can be tested using the included client. Two tests should be performed.

Test One

The included client is located in the Node project (.jpr) inside JDeveloper.

1. Open JDeveloper and the Node.jpr folder.
2. Open the Miscellaneous Files folder and the file called Locator.properties.
3. Make the following change: comment out the CDX URL by placing a # symbol at the front of the line.
4. Add the following line: the URL to which you deployed the Node (above). For this use the non-SSL port number, as you are probably behind your firewall. JDeveloper doesn't natively understand the SSL configuration used by the CDX DNC.
5. Inside Client.java (gov.epa.state.sample.client) change the variables username and password for the CDX node user account you created.
6. Save your changes in JDeveloper.
7. Go to the package gov.epa.sample.client
8. Right click on Client.java and select Run. The file will compile automatically before it is run.

The change to the Locator.Properties file above will cause the client to run against your deployed Node. The output console for the run will show the log of the SOAP requests and responses as well as client logs.

If everything works correctly, you should see the following lines appear at the end of the run:

```
*** [DEBUG][When: 2003/10/03_13:21:57:468] Where: gov.epa.state.sample.client.Client
    Message: [getServices]: End of List of supported services.
************************************************************

Process exited with exit code 0.
```

Test Two

From test one above take the transactionId returned from the Solicit method call. Open the file "developer_scripts.sql" located at: \DNC\Source_Code\database\PL_SQL. Copy the transaction Id into the second script (mentions the Node making a call to itself), follow the comments. Start a SQL*Plus session, logging in as the schema user. Copy and paste this script into the session and run it.

When the Cron job runs in the background, the .jar file will call your Node's Submit method. It will pass in the XML document that was created as a run of the DNC client. This will test the back end processes to make sure that a document is generated and sent (pushed) as a result of a call to the Solicit method.

To test with the Federal test Web site:

Each Node must have administrator and operator accounts set up in order to differentiate its transactions from those of other Nodes and to identify it for Network security purposes. To create a Node administrator account for the Exchange Network, contact the CDX Help Desk at Nodehelpdesk@csc.com or 1-888-890-1995. **Do not use this administrator account for testing the node. Create a separate operator account through the CDX Help Desk for testing.**

If your State is ready to put its Node into testing with the Federal Node testing Web site:

Contact the Exchange Network Help Desk to establish an account. Provide the following information: Full Name, Mail Address, Phone Number, E-mail Address. They will provide you with a user name and password. Even if you have an existing CDX account, you'll need to call the help desk so your accounts can be synchronized.

If your State Node is ready to go into production:

Please contact the CDX Help Desk to establish a production account so that your production data will not be submitted to the Test Node. The Help Desk can assist you in establishing both administrator and user privileges for Node test or production accounts.

Provided you have an account (mentioned above), the current test or "Integration" tool is located at:
https://test.epacdxNode.net/test/index.jsp

1. Click on the link Authenticate, under Initialization.
2. This will bring up the Authenticate page, enter your account information. Make sure that the Node Address is that of your published Node.
3. Click on the Invoke button. This should authenticate you with your Node.
4. Under Automatic Tests, click on the link All Services.
5. On the next screen click on the Invoke button.
6. This will begin a test of all the services on your Node. A screen will appear at the end of testing to show the results. You can click on the links beside the method to get information to help troubleshoot errors.

Troubleshooting (or FAQs)**:**

Here are a few potential problems that may come up while implementing the Node, and possible solutions for them.

1. **Error**:
   "tns-service name not found"

   Possible solution: Could be due to your code calling the wrong schema/database name. Another possible solution is that the sqlnet.ora file is missing from the application server (check with your Administrator).

2. **Error**:
   ```
   <faultcode>soapenv:Server.userException</faultcode>
   <faultstring>java.lang.NullPointerException</faultstring>
   ```

   Possible solution: Could be caused by the fact that the application can't see the user_properties.XML file. That file actually describes the implementation class for the Node (find it under ..\DNC\Source_Code\Node\Node\conf\properties directory). If the file "user_properties.XML" is not under the "home" of the application server, the application will not run. See the "Installing the Application Server" section of the DNC to configure this.

3. **Error**:
   ```
   ORA-25207: enqueue failed, queue DEPNODE.SOLICIT_JOB_QUEUE is disabled from
   enqueueing ORA-06512: at "SYS.DBMS_AQ", line 204 ORA-06512: at
   "DEPNODE.QUEUEADMIN", line 47 ORA-06512: at
   "DEPNODE.INCOMING_SOLICIT_TRIGGER", line 3 ORA-04088: error during
   execution of trigger 'DEPNODE.INCOMING_SOLICIT_TRIGGER'
   ```

   Possible solution: May be caused when the queue table is disabled for enqueueing from the Oracle Advanced Queueing. Two possible causes are:

   1) The queue is stopped and needs to be started. Login as SYSTEM and run this script:

      ```
      begin
      dbms_aqadm.start_queue
      (queue_name     => ('SOLICIT_JOB_QUEUE');
      end;
      /
      ```

   2) The permissions aren't in place to invoke the queue. Login as SYSTEM and run this script:

      ```
      grant aq_administrator_role  to  <Your_Schema>;
      ```

4. **Error**:
   ```
   [SOAPException: faultCode=SOAP-ENV:Client; msg=No Deserializer found to
   deserialize a
   &apos;http://schemas.XMLsoap.org/soap/envelope/:Parameter&apos;
   using encoding style
   &apos;http://schema.XMLsoap.org/soap/encoding/&apos;.;
   ```

   Possible solution: This is caused by the Node sending the response. The value for the encodingStyle attribute in the response is invalid. The "s" is missing in the URL (should be schemas.XMLsoap.org/soap/encoding/).

Issues/Lessons learned

1. The developer of the Node should have the following skill set:
   a. Java, J2EE, PL/SQL
   b. Oracle 9iAS J2EE deployment
   c. Apache Axis
   d. JDeveloper
   e. UNIX or Cron knowledge
   f. Web Services knowledge
2. Web Services standards keep evolving; many companies submit different standards for doing the same thing. For an example see the Oracle document attachment at the end of this document.
3. It's relatively easy to use Oracle J2EE Web services to develop the Web services and Web services client. Integrating with other toolsets, when elements such as a standard WSDL file is required, will require additional work.
4. Things that Oracle currently can not support (pre-10g Application Server v9.0.4) :
   a. DIME protocol
   b. FaultDetail can not be updated in SOAP exception fault message.
   c. Mixed Encoding Styles in a SOAP message body vs. SOAP header

   Answer: To resolve A and B above, the solution was to use Apache Axis by using the CSC DNC as a wrapper around Maine's code. To resolve C above, the Functional Specification v1.0 was changed so mixed encoding is not required.
5. JDeveloper 9.0.3 does not directly support use of the HTTPS protocol.

   Answer: We have built in a work-around for the client.
6. The Node 1.1 specification details the methods "exposed" to the client, but provides no detail specification on the internal processing to "connect" those methods together.

   Answer: Oracle created the Scheduler (PL/SQL) procedure as one of the PL/SQL procedures to handle SOLICIT, SUBMIT and DOWNLOAD requests.
7. ANT can integrate with JDeveloper for deployment of Apache Axis. Using an ANT build script, Apache Axis was integrated with the Node to deploy to Oracle 9iAS.
8. Since many toolkits cannot accept a null value, parameters rowId and maxRows are not optional. This problem arises because of different interpretation of SOAP specifications by different toolkit vendors.

   Answer: Query method parameters rowId must be 0 and maxRows must be -1 if a positioned fetch is not requested.
9. For compatibility, it is better to keep datatypes of the "standard" types available to exchange with SOAP. Custom datatypes can cause additional work depending upon the platform implementing them.
10. Nodes MUST implement the network official WSDL file http://test.epacdxNode.net/schema/v1.0/cdx.wsdl.
11. The WSDL generated by JDeveloper 9.0.3 does not reflect the official network WSDL. In particular, the values of message name attributes did not match. To give an example, in JDeveloper the name was NodePing0Request and the official WSDL calls that NodePing. For NodePing response, the official name is NodePingResponse. JDeveloper called it NodePing0Response. There was also an issue with Oracle using int for Integer.

    Answer: We used the CSC DNC with JDeveloper.
12. Oracle does not believe there's any issue to running the DNC based on an Oracle 9.1 database. Regarding the Oracle 8.1.7 database, the XSU (generates XML from the database) might be an issue. However, the guess is 99.9 % should be OK. Maine based the development on an Oracle 9.2 database.

13. Some statements in the Protocol and Functional Specifications are forward-looking but are not identified as such. Even though a Node doesn't need to handle Event or Status notification, the easiest way to implement the flow handling in the Notify method is to check whether the dataflow parameter is one of the URLs below. If not, the message is a document notification (all other values are real dataflow identifiers, e.g., FRS, NEI, etc.).

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

14. When leveraging the CDX DNC as the wrapper for the Maine Node, the method signature for the Download method shows the return datatype of void. In the method signature, the parameter ArraryofDocHolder handles returning the array of Node documents back to the caller. Here are a few comments from CSC:

    - For the Download method: gov.epa.cdx.axis.v10.vo.holders.ArrayofDocHolder is a class that Axis needs. It is simply a holder (wrapper) for the array of Node Documents. You can set the array of documents into the ArrayofDocHolder object using its constructor:
      public ArrayofDocHolder(gov.epa.cdx.axis.v10.vo.NodeDocument[] value).
    - The only place where the holder is used, is download method -- which is the only method that returns documents to the callee. So you can return documents to the callee by setting array of documents into download method's argument.

15. Once we began using the CDX DNC we found that you have to convert your exceptions into SoapFault in order to throw them properly according to the spec. Additionally, you need to create some extra fields inside the AxisFault in order to be compliant with our specification. Throwing a regular exception will cause the toolkit to try to format the AxisFault the best way it can (not a good idea).

    CSC has created some helper methods which will help you convert Exceptions into AxisFault messages that comply with Node Functional Specifications v1.1. These methods are accessed by:

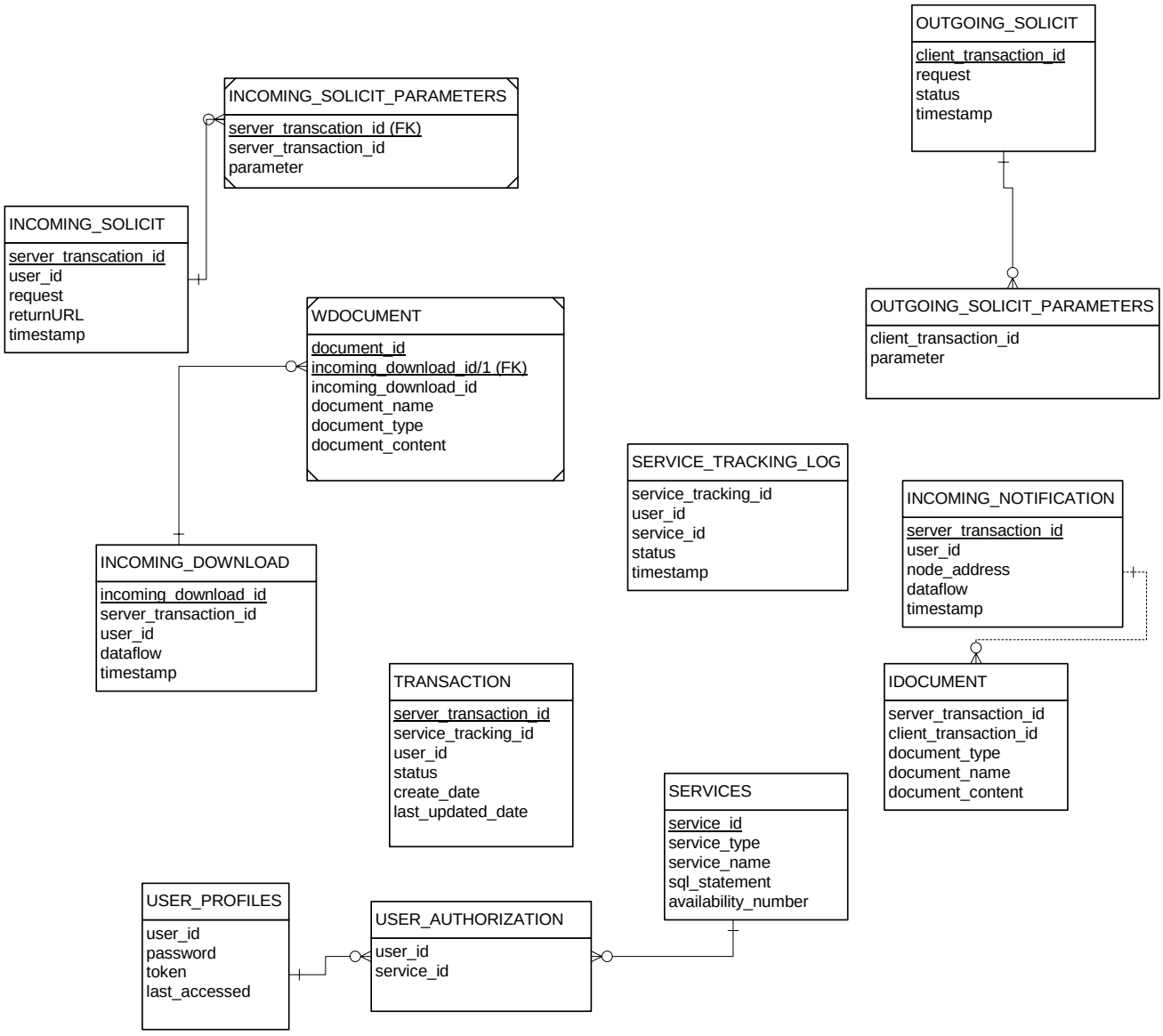
    AxisFault fault = FaultHandler.getFault("Client", "Login Failure", FaultErrorCode.UNKNOWN_USER)

    The first parameter is the fault code string, the second is the fault message, and the third is the fault error code. There is a list of canned FaultErrorCodes that are compliant with the specification; look at FaultErrorCode's static variables.

16. Maine uses a Verisign SSL certificate. When testing with the Federal Integration tool, WebLogic had a problem with Verisign Extensions in the certificate. This has been fixed with the installation of Weblogic 7.0 Service Pack 4.
17. JDeveloper 9.0.3 was not used to deploy directly to the application server. When setting up the connection to the server (using the connection wizard), the administrator account must be used. During the testing of the connection, the account password is displayed unencrypted on the log file. Avoiding this security issue is the reason for using JDeveloper to deploy an .ear file which is later sent to the administrator of the application server.
18. In using JDeveloper 9.03, the Web server port number at which the service could be found was declared in the Web Services wizard. When utilizing Apache Axis, Axis is able to handle requests on other port numbers, so changes in port numbers should be handled behind the scenes (no changes by the developer).

Database Table Design

E-R Diagram



Database Tables

Table: USER_PROFILES

Stores authenticated user information.

Attributes	Type	Description
User_id	VARCHAR2	User id
Password	VARCHAR2	password
Token	VARCHAR2	This field will be updated with the token value, the token is generated by a centralized authentication system when user is authorized for this Node Server.
LAST_ACCESSED	DATE	TimeStamp of last access time.

Table: USER_AUTHORIZATION

Matches a user_id with the services they are authorized to use.

Attributes	Type	Description
User_id	VARCHAR2	User id
Service_id	VARCHAR2	Service id – each service in the Node Server have a service id. Service_id is the key to the SERVICES Table

Table: SERVICES

Stores a list of all services available from this Node.

Attributes	Type	Description
Service_id	VARCHAR2	Service id
Service_type	VARCHAR2	Service type – 'INTERFACE', 'QUERY'
Service_name	VARCHAR2	Service Name-

		'AUTH','QUERY','EXECUTE', 'SUBMIT' ….
SQL_Statement	VARCHAR2	SQL Statement SQL Statement for this service if the Service_type = 'QUERY'
Availability_number	NUMBER	Availability number Service available or not Null - available

Table: TRANSACTION

When client invokes SUBMIT, SOLICIT, or NOTIFY, Node Server will return a transaction Id, and the record is inserted into this table.

Attributes	Type	Description
Server_transaction_id	VARCHAR2	Transaction id for each transaction
Service_tracking_id	VARCHAR2	Service tracking id
User_id	VARCHAR2	User id
Status	VARCHAR2	Status of the transaction
Create_date	Date	Creation date
Last_update_date	DATE	Last update date

Table: SERVICE_TRACKING_LOG

Used to log which services a user accesses, if the service was successfully executed, and the time a service was accessed.

Attributes	Type	Description
Service_tracking_id	VARCHAR2	Service tracking id
User_id	VARCHAR2	User id
Service_id	VARCHAR2	Service id of the service client access
Status	VARCHAR2	Current status

Timestamp	DATE	Timestamp

Table: INCOMING_SOLICIT

Stores Solicit requests from a client. Logs the user, query requested, and a transactionId to match the parameters stored in the INCOMING_SOLICIT_PARAMETERS table.

Attributes	Type	Description
Server_transaction_id	VARCHAR2	Service transaction id
User_id	VARCHAR2	User id
Request	VARCHAR2	Service Request
Return_URL	VARCHAR2	After Node Server processed this request, the return_URL will be used to post the result back (SUBMIT) to Client
Timestamp	DATE	Timestamp

Table: INCOMING_SOLICIT_PARAMETERS

Stores parameters for the incoming SOLICIT call. The transactionId is used to match each parameter to the INCOMING_SOLICIT query in the INCOMING_SOLICIT table.

Attributes	Type	Description
Server_transaction_id	VARCHAR2	Service transaction id
Parameter	VARCHAR2	Parameter values for the request's SQL query.

After the Node server processes the SOLICIT request, the result (XML document) will be stored in INCOMING_DOWNLOAD and WDOCUMENTS tables for the client to download later.

Table: INCOMING_DOWNLOAD

Attributes	Type	Description
Server_transaction_id	VARCHAR2	Server transaction id
Incoming_download_id	VARCHAR2	Download id

User_id	VARCHAR2	User id
Data_flow	VARCHAR2	Data flow
Timestamp	Date	Timestamp

Table: WDOCUMENTS

Attributes	Type	Description
Incoming_download_id	VARCHAR2	Download id
Document_id	VARCHAR2	Document id
Document_name	VARCHAR2	Document Name
Document_type	VARCHAR2	Document Type
Document_content	VARCHAR2	Document Content

Table: OUTGOING_SOLICIT

Stores the request data when the Node Server acts as a client and invokes SOLICIT request on other Node Server.

Attributes	Type	Description
Client_transacton_id	VARCHAR2	Transaction id that the Node get when the Node invoke solicit request of another Node Server, that Node server will return a transaction id for the SOLICIT request.
request	VARCHAR2	Name of QUERY
Status	VARCHAR2	Status of the SOLICIT request
Timestamp	DATE	timestamp
Document_content	VARCHAR2	Document Content

Table: OUTGOING_SOLICIT_PARAMTERS

Attributes	Type	Description
Client_transaction_id	VARCHAR2	Transaction id in OUTGOING_SOLICIT tables.
Parameters	VARCHAR2	Value of the parameter for this solicit request.

Table: INCOMING_NOTIFICATION

When client notifies the Node Server by using NOTIFY request, information will be stored in INCOMING_NOTIFICATION and IDOCUMENT tables.

Attributes	Type	Description
Server_transaction_id	VARCHAR2	Node Server will return a transaction id to the client who notifies the Node Server. Node Server also stores the transaction id in this table.
User_id	VARCHAR2	User id of the client who notify the Node Server
Node Address	VARCHAR2	Node address of the client if the client itself is also a Node Server, otherwise – null.
Dataflow	VARCHAR2	Dataflow
TimeStamp	DATE	Timestamp

Table: IDOCUMENT

Attributes	Type	Description
Server_transaction_id	VARCHAR2	Node Server will return a transaction id to the client who submits a document

		to the Node Server. System stores the transaction id in this table.
Client_transaction_id	VARCHAR2	The corresponding SOLICIT request made by the Node Server, the transaction id is the id in OUTGONG_SOLICIT table.
Document_type	VARCHAR2	Document type
Document_name	VARCHAR2	Document name
Document_content	VARCHAR2	Document Content

Advanced Queues

Two Advanced Queues are created for the Maine Node. These are used to do the asynchronous processing on calls to the Solicit method.

Queue Name

SOLICIT_JOB_QUEUE

Queue Table Name

SOLICIT_JOB_QUEUE

Payload Type

JOB_TYPE

Queue Usage

The entire Solicit request from the clients will be enqueued in the Solicit Queue. A Scheduler (PL/SQL) acts as a background job to dequeue the data from this queue and process it.

Queue Name

SUBMIT_JOB_QUEUE

Queue Table Name

SUBMIT_JOB_QUEUE

Payload Type

JOB_TYPE

Queue Usage

The Node Server is a client, and all the requests which are required to call SUBMIT on other Node Servers will be enqueued in the SUBMIT_JOB_QUEUE. A listener will dequeue the data and act as a client to call the corresponding Node Server's SUBMIT to submit the data.

PL/SQL to create the PAYLOAD TYPE

```
CREATE OR REPLACE TYPE JOB_TYPE AS OBJECT

(server_transaction_id VARCHAR2(100));

/
```

PL/SQL to create the AQ

```
CREATE OR REPLACE PACKAGE BODY queueAdmin AS
```

```
     PROCEDURE createQues(
       dbusname                    IN     VARCHAR2,
       quetablename                IN     VARCHAR2) IS

    quename    VARCHAR2(100);
BEGIN

    quename   := dbusname;

    DBMS_AQADM.CREATE_QUEUE(
           Queue_name          => quename,
           Queue_table         => quetablename,
           max_retries         => '2');


    DBMS_AQADM.START_QUEUE(
           queue_name         => quename);
    END;


    PROCEDURE createQueTable(quetablename IN VARCHAR2) IS
    BEGIN
           DBMS_AQADM.CREATE_QUEUE_TABLE(
                  Queue_table            => quetablename,
                  Queue_payload_type     => 'JOB_TYPE',
                   multiple_consumers     => false,
                  compatible             => '8.1.5');

    END;

    PROCEDURE enqueueQue(v_id IN VARCHAR2, queueName IN VARCHAR2) IS

    Enqueue_options     DBMS_AQ.enqueue_options_t;
    Message_properties  DBMS_AQ.message_properties_t;
    Message_handle      RAW(16);
    Message             job_type;
    BEGIN
           Message := job_type (v_id);

           DBMS_AQ.ENQUEUE(queue_name => queuename,
                  Enqueue_options            => enqueue_options,
                  Message_properties         => message_properties,
                  Payload                    => message,
                  Msgid                      => message_handle);

    END;
END  queueAdmin;
/

EXEC QUEUEADMIN.createQueTable('SUBMIT_JOB_QUEUE');
EXEC QUEUEADMIN.CREATEQUETABLE('SOLICIT_JOB_QUEUE');
EXEC QUEUEADMIN.CREATEQUES('SUBMIT_JOB_QUEUE','SUBMIT_JOB_QUEUE');
EXEC QUEUEADMIN.CREATEQUES('SOLICIT_JOB_QUEUE','SOLICIT_JOB_QUEUE');
```

Stored Procedures

The following PL/SQL procedures are created

Package name: SQLQUERY

Procedure

GETFACILITYBYNAME (STATEID IN VARCHAR2, FAC_NAME IN VARCHAR2, FROM_ROW_NO IN NUMBER, ROW_NO IN NUMBER, RES_OUT OUT CLOB)

Program Description

This procedure is used for the QUERY method. It selects the records from the DEPNODE table with the following arguments:

Parameter Description

FAC_NAME – facility name

FROM_ROW_NO – start row number

ROW_NO – total number of records to return

RES_OUT – XML document of the result

Procedure

GETFACILITYBYLOCALITYNAME (STATEID IN VARCHAR2, LOC_NAME IN VARCHAR2, FROM_ROW_NO IN NUMBER, ROW_NO IN NUMBER, RES_OUT OUT CLOB)

Program Description

This procedure is used for the QUERY method. It selects the records from the DEPNODE table with the following arguments:

Parameter Description

LOC_NAME – locality name

FROM_ROW_NO – starting row number

ROW_NO – total number of records to return

RES_OUT – XML document of the result

Procedure

GETFACILITYBYSTATEID (STATEID IN VARCHAR2, FROM_ROW_NO IN NUMBER, ROW_NO IN NUMBER, RES_OUT OUT CLOB)

Program Description

This procedure is used for the QUERY method. It selects the records from the DEPNODE table with the following arguments:

Parameter Description

STATEID - state id

FROM_ROW_NO – starting row number.

ROW_NO – total number of records to return

ROW_NO – total number of records to return

RES_OUT – XML document of the result

Procedure

GETFACILITYBYENVID (STATEID IN VARCHAR2, ENVR IN VARCHAR2, FROM_ROW_NO IN NUMBER, ROW_NO IN NUMBER, RES_OUT OUT CLOB)

Program Description

This procedure is used for the QUERY method. It selects the records from the DEPNODE table with the following arguments:

Parameter Description

ENVR – environmental

FROM_ROW_NO – staring row number to retrieve the records

ROW_NO – total number of records to return

RES_OUT – XML document of the result

Procedure

GETFACILITYBYCHANGEDATE (STATEID IN VARCHAR2, VDATE IN VARCHAR, FROM_ROW_NO IN NUMBER, ROW_NO IN NUMBER, RES_OUT OUT CLOB)

Program Description

This procedure is used for the SQLQUERY method. It selects the records from the DEPNODE table with the following arguments:

Parameter Description

VDATE – change date

FROM_ROW_NO – starting row number to retrieve the records

ROW_NO – total number of records to return

RES_OUT – XML document of the result

Package Name: USER_AUTH

Procedure

GET_USER (ITOKEN IN VARCHAR2, IUSER OUT VARCHAR2)

Program Description

This procedure is used to retrieve the username by using the token.

Parameter Description

ITOKEN – token

IUSER - username

Procedure

AUTHORIZE (IUSERID IN VARCHAR2, SERVICE_ID IN VARCHAR2, RETURN_CODE OUT NUMBER)

Program Description

This procedure checks if the user is authorized to access that Node service. Service id and service name are stored in the SERVICES table.

Parameter Description

IUSERID – username

SERVICE_ID – service id

RETURN CODE - 0 – authorized, 1 – not authorized

Package Name: SCHEDULER

Functional Steps

The following outlines the functionality that the Scheduler performs:

1. Dequeue from SOLICIT_JOB_QUEUE queue.
2. Retrieve the server_transaction_id from the payload.
3. Select request, returnURL from incoming_solicit table.

4. Select sql_statement from services for that request.
5. Select parameters from incoming_solicit_parameters.
6. Create the query and run the query.
7. Format the result as XML document.
8. If URL is not null, enqueue XML document, server_transaction_id, dataflow to outgoing_submit_queue.
9. Update the transaction status = "processed".
10. Save it in the database system.
    a. Write to incoming_download table with:
       User_id (client)
       Server_transaction_id
       Dataflow
    b. Write to wdocument table with:
       Document_type
       Document_name
       Document_content

Procedure

RUN (SOLICIT_QUEUE IN VARCHAR2)

Program Description

This procedure dequeues the data from the SOLCIT_QUEUE. The payload of the queue is an object, which has a transaction id. Scheduler uses the transaction id to retrieve the related transaction information (request name and request parameters) from INCOMING_SOLICIT and INCOMING_SOLICIT_PARAMETERS tables. It executes the request and creates an XML document. After that, it enqueues the transaction id in the SUMBIT_JOB_QUEUE, and it inserts a new record in INCOMING_DOWNLOAD table, and inserts the output XML document (CLOB) into the WDOCUMENT table. **STOP Procedure must be run before RUN Procedure to place the stop transactionId in the** SUBMIT_JOB_QUEUE **queue.**

Parameter Description

SOLICIT_QUEUE – queue name of the SOLICIT queue in the system.

Procedure

STOP (SOLICIT_QUEUE VARCHAR2)

Program Description

This procedure will enqueue a "flag" (an invalid transaction id) into the SOLCIT_QUEUE, which will cause the Scheduler to stop to dequeue message. **STOP Procedure must be executed before RUN Procedure to place the stop transactionId in the** SUBMIT_JOB_QUEUE **queue.**

Parameter Description

SOLICIT_QUEUE – queue name of the SOLICIT queue in the system

Remark

The records in INCOMING_DOWNLOAD, WDOCUMENT and SUBMIT_JOB_QUEUE are used for the Node Server to submit the results back to the client. Included with the Node code is a Java client

program (SubmitListener.java). This program dequeues the transaction id from SUBMIT_JOB_QUEUE and retrieves the related records from the INCOMING_DOWLOAD and WDOCUMENT tables using the transaction id. The client then retrieves the returnURL from the INCOMING_SOLICIT table by using the transaction id. After getting all of the information, the client program invokes the SUBMIT method to send the document back to the requestor (other Node).

Program Description

GetStatus

Database Tables: transaction, service_tracking_log

Description:
GetStatus will retrieve the status field from the Transaction table.

Service tracking will be written to the service_tracking_log table.

Solicit

Database Tables: incoming_solicit, incoming_solicit_parameters, transaction, service_Tracking_log

AQ: solicit_job_queue

Description:
When a client sends a SOLICIT request, the request and the parameter array will be written into incoming_solicit and incoming_solicit_parameters tables, a transaction record is created in transaction table with the status = 'received', and the transaction id is returned to the client.

A trigger in the incoming solicit will be fired to enqueue the transaction id to the solicit_job_queue.

Service tracking will be written to the service_tracking_log table.

Submit

Database Tables: idocument, transaction, service_tracking_log

Description:
When a client invokes a submit request, the document will be saved in the idocument table, a transaction record is created in the transaction table with the stats = 'received', and the transaction id is returned to the client.

Service tracking will be written to the service_tracking_log table.

Download

Database Tables: incoming_download, wdocument, service_tracking_log

Description:
Download records and documents have to exist in the incoming_download and wdocument tables for client to download. Client invokes download to download document.

If the transaction id is not null in the request, the system will retrieve the incoming download record from the incoming_download table by using the transaction id, then retrieve all the documents in wdocument table with that transaction id and return the documents to client.

If transaction id is null, system will retrieve the documents from the wdocument & incoming_download tables with matching userId, dataflow, document name, and document type.

Service tracking will be written to the service_tracking_log table.

Notify

Database Tables: incoming_notify, idocument, transaction, service_tracking_log, transaction

Description:

A client invokes the notify method to inform a Node of a change in status or that a document is ready for retrieval or event (such as CDX Node briefly unavailable while server patches are put into place).

When a client invokes the notify request, the notification information will be written to the incoming_notify and idocument tables.

A transaction id is generated, and a transaction record is added to the transaction table. The transaction id is returned to client.

Service tracking will be written to the service_tracking_log table.

Query

Database Tables: service_tracking_log, services

Description:
A client invokes the query request. By using the request in input, the system will retrieve the SQL statement from services table. The system will prepare a SQL statement with parameters to be executed in JDBC. The corresponding PL/SQL procedure will be executed in the database, and the result will be returned as a CLOB (XML document). System returns the result (XML document) as a String.

Service tracking will be written to the service_tracking_log table.

GetServices

Database Tables: services

Description:
A client invokes the GetServices request.

By using this request, the system will return the services available on the Node Server.

NodePing

Description:
A client invokes the Ping request.

By using this request, the system will return the availability of Node Server. Node Server's availability is set by a static variable in the program.

Authenticate

Description:
A client invokes this method and passes in userId, password, and authentication method type.

The Node uses NAAS (Network Authentication and Authorization Service) or central authentication to verify that the user's information is valid. Once the user is approved to use the system, NAAS will send back a token.

The token sent by NAAS will be returned to the client requesting authentication. The returned token will be a required parameter in all other methods the Node offers.

Service tracking will be written to the service_tracking_log table.

SSL Client

The following information is for developers who wish to better understand how the Submit client is built. By changing the connection information in the SubmitListener.java file and deploying the .jar file above under "Prepare Deployment files" above, these steps have already been taken.

Maine has implemented a client using JDeveloper. The code is included in the DNC under the JDeveloper project "SubmitClient.jpr". The client uses a UNIX job scheduler ("cron") to wake up and process all solicit requests in the queue. The following parts must be assembled in order for this client to run:

- Java Code (3 classes). NetworkNodeStub.java, SubmitListener.java, and NodeDocument.java
- PL/SQL code compiled in the database (is in the .dmp file the DBA imported while setting up the database above).
- Cron job and supporting scripts placed in the application server (should have been done in the Install and Setup the Application Server above).
- Oracle Wallet exported and placed in a folder on the application server for the Web Services "stub" NetworkNodeStub.java to access. This should have been done under the Install and Setup of the Application Server documentation above.

Prior to the Java client executing, the Cron (job scheduler) wakes up and executes three PL/SQL procedures.

- execute SCHEDULER.STOP('SOLICIT_JOB_QUEUE') : This places a "stop" flag in the Solicit_Job_Queue.
- execute SCHEDULER.RUN('SOLICIT_JOB_QUEUE') : This processes the solicit calls and drops the XML documents into the corresponding tables (ready for pick up).
- execute STOP_SUBMITLISTENER('SUBMIT_JOB_QUEUE') : This places a "stop" flag in the Submit_Job_Queue, so that the Java Client will stop running.

Next, Cron executes the .jar file. This will cause the Java client to attempt a call to the Node that originated the Solicit request. This is a single attempt. The status of the document should have been updated when SCHEDULER.RUN was executed above, and so the "callee" Node will have to use the Download method if this attempt is unsuccessful.

The prior two paragraphs currently execute (Cron runs) on the hour and half-hour. A state may change this if needed to run less or more frequently. An explanation of how the Java portion of the client works follows.

First, the Java code is written to retrieve the information from the database on documents that have been placed in the SUBMIT_JOB_QUEUE. After passing business edits, the code instantiates a Web service stub handing in the URL retrieved from the database. The stub should provide a handle to the Node who called the Solicit method on your Node. The client then attempts to Authenticate itself, after which it calls the Submit method attempting to pass the NodeDocument(s) that it retrieved from the database.

In order for the SSL Web service stub to work, the Oracle Wallet must be in a folder that is accessible to it. When the JDeveloper 9.0.3 wizard for building Web Service Stub/Skeleton is used to build the stub for accessing other Nodes, the code generated requires some modifications. View the code snippet below and modify the wizard generated stub as needed:

```
public NetworkNodeStub(String returnURL)//String returnURL
{
  m_httpConnection = new OracleSOAPHTTPConnection();
```

```
    m_smr = new SOAPMappingRegistry();

    BeanSerializer beanSer = new BeanSerializer();
    m_smr.mapTypes(Constants.NS_URI_SOAP_ENC, new
QName("http://www.ExchangeNetwork.net/schema/v1.0/Node.xsd", "NodeDocument"),
gov.maine.Node.client.NodeDocument.class, beanSer, beanSer);

    // ********************************************************  //
    // The next few lines are what makes SSL work for this Client
    System.setProperty("ssl.SocketFactory.provider",
"oracle.security.ssl.OracleSSLSocketFactoryImpl");
    System.setProperty("ssl.ServerSocketFactory.provider",
"oracle.security.ssl.OracleSSLServerSocketFactoryImpl");
    System.setProperty("java.protocol.handler.pkgs","HTTPClient");

System.setProperty("oracle.wallet.location","<application_server_location_of_O
racle_wallet>");
    System.setProperty("oracle.wallet.password","<your_wallet_password>");
    // End of SSL code
    // ********************************************************  //

    //switch the endpoint
    endpoint = returnURL;
  }

  public String endpoint =
"https://test.epacdxNode.net/cdx/services/NetworkNodePortType_V10";
  private OracleSOAPHTTPConnection m_httpConnection = null;
  private SOAPMappingRegistry m_smr = null;
```

Basically, you need to use System.setProperty to set those properties, and modify the endpoint. For the property "oracle.wallet.location", please put the location you saved the "exported wallet". Additionally, the following libraries are required (you can copy them from the iAS machine):

$ORACLE_HOME/lib/jsse.jar
$ORACLE_HOME/jlib/jssl-1_2.jar
$ORACLE_HOME/jdk/jre/lib/ext/jcert.jar

You can build a single library of the three .jar files. Include it in your project. Use the buttons on the window to make it the first library on the list.

Other SSL Information

A question was brought up as to whether a Node should implement "mutual authentication" using SSL. Here the answer that we received from CSC:

Mutual authentication is more complicated than it sounds. I would not recommend the use of SSLVerifyClient unless absolutely required. Here are the reasons:

1. Mutual authentication requires that clients to have a valid SSL certificate issued by a trusted certificate authority (CA). Since getting an SSL certificate is not easy and the certificate costs money (plus annual renewals), it is often not feasible to have mutual authentication for a large group of users. The reason SSL is so popular in the e-commerce world is largely because it doesn't require clients to have an SSL certificate, and only SSL server authentication may be necessary (a commercial site trusts all users who can send money to it).
2. Unlike traditional user accounts where you can delete an account when the user is no longer with an organization, you can't get the certificate back once issued. Maintaining and managing certificates for mutual authentication require very complicated PKI in place.
3. In a situation where user may use any certificates issued by commercial CAs, mutual authentication is often problematic because you don't know how many CAs you have to trust, and whether or not a CA is really trustworthy.

Even though mutual authentication with SSL can be implemented, it is not suitable and insufficient for many Web service applications, such as network Nodes. Remember SSL authentication is on the transport layer, a Web service application would have a hard time figuring out who has logged in and what privileges the user has.

However, authentication using a certificate on the application layer is quite feasible and suitable for Web services. We are currently looking into using XML signature with certificate for user authentications on NAAS, assuming we can have our own CA and be able to issue certificates free of charge to many users.

Here are two document IDs that can be referenced in Oracle Metalink for those who might want additional information:

184432.1	Configuring SSL with Oracle HTTP Server in 9iAS
184433.1	Configuring 9iAS Release 2 SSL Client

Other Options

This section lays out another possible option that a state might take in developing their Node. This section is more of a "hints" section as the Maine DNC did not follow the possible paths below.

SSL Client

An SSL Client can be created as a sample code by using Sun's Java API.

1. DummyTrustManager implemented TrustManager. This dummy TrustManager will approve any certificate.

2. DummySSLSocketFactory implemented SSLSocketFactory. This DummySSLSocketFactory uses DummyTrustManager as its Trust Manager.

3. Client.java use DummySSLSocketFactory to invoke SSL call to other Web services.

4. It is only a sample code, since the DummyTrustManager will just approve any certificate.

5. A valid TrustManager class has to be used for production.

Log4J notes (from CSC)

1. There should be a properties file cdxLog.properties at the folder path: ..\DNC\Source_Code\Node\Node\ at the same level in the hierarchy as the build.XML file. This file should configure things so that the application can use log4J, to for logging of messages.
2. On the Application Server, place cdxLog.properties under /j2ee/home directory (the location sounds weird, but the file should be placed there, not under /j2ee/yourinstance/)
3. Go into Enterprise Manager. Select the instance where your Node application is deployed. Click on Server Properties.
4. OC4J Options text box should have the following there: -properties -out oc4j.out -err oc4j.err (if not there copy and paste into the text box)
5. Click apply
6. Restart OC4J instance.

After application server is restarted and the test tool is run against it. You should see Node.log4j file under /j2ee/home directory It will contains all the log statements.

Also you will see a directory called {your_instance_name}_default_island_1 under /j2ee/home inside there you will find 2 files oc4j.out and oc4j.err - this is where all System.out and System.err printout statements will go. You should stay away from System.out or System.err, it has large performance consequences and you will have to go through your code and comment out System.out before going to production.

One of the benefits of using the log4j way of logging is that when you go into production, you can simply modify cdxLog.properties not to log - it will stop logging automatically, without any code changes. Another benefit is that logging is done in buffered fashion and without blocking the main thread of execution.

Deployment Models

Oracle J2EE Web Services uses a tiered architecture and can be deployed in a number of different ways.

This section describes the different options. Choose the deployment model that best suits your requirements

Deployment Model #1

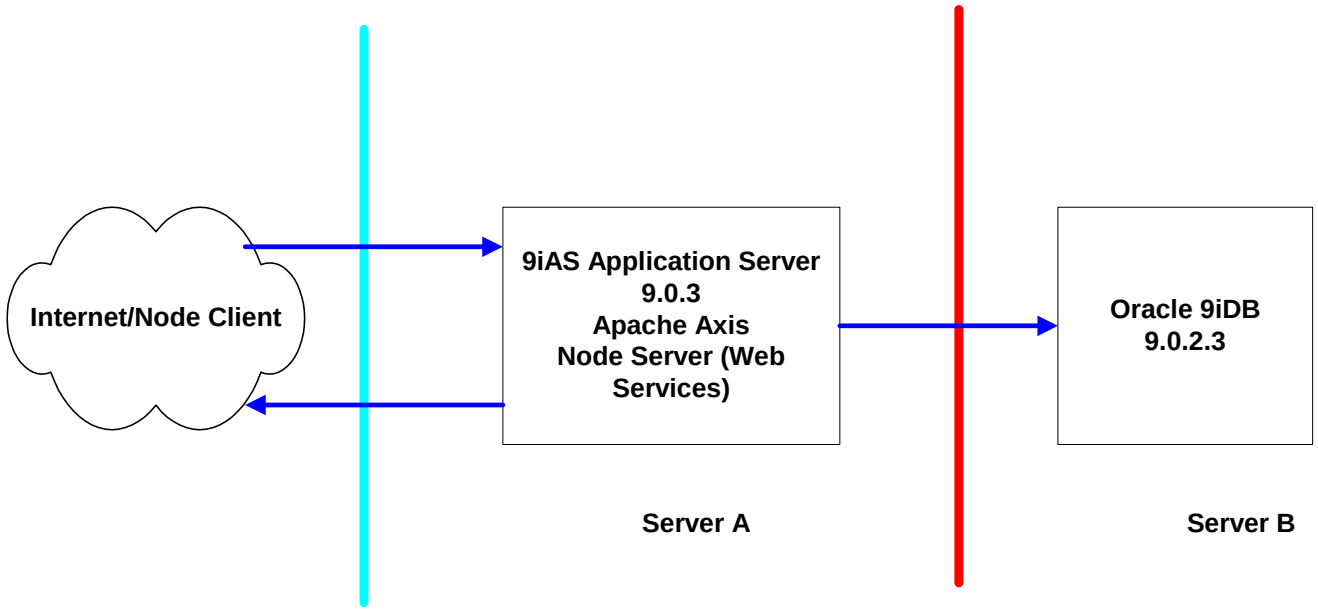
This is the simplest of the deployment models and has all the components installed on the same server. **Note: Maine never used this as a possible option.*

Component Distribution

All components are installed on a single server.

Deployment Model #2 (Recommended)

This is the classic 2-tier model where only the Internet facing components are installed in a MZ (a.k.a. DMZ) and all other components are setup on a trusted, secure network



Distribution

- **Server A**
  - Oracle 9i Application Server
  - Apache Axis
  - Node Server (Web Services)
- **Server B**
  - Oracle 9iDB (PL/SQL, Database Objects)

There are several ways that one can use JDeveloper to deploy Web Services

1. It can be placed in the same machine as the server, then an "application server connection" can be built using the wizards in JDeveloper by which the Web services can be deployed (to the server on the same machine). The connection is 'Application Server - local DCM'

2. Option two is that JDeveloper 9.0.3 can be used to connect to a remote box. The problem with this option is that the server must also be at version 9.0.3. An additional issue here is that when the connection is tested the password (which is encrypted in the setup) is shown to the developer. The JDeveloper 9.0.3 can connect to 9.0.3 or 9.0.2.x, but for 9.0.2.x, a servlet (DCMAPI Servlet) has to be deployed. Review the documentation <JDeveloper_HOME>\jdev\readme.html that comes with JDeveloper (Installing the DCM Servlet to Deploy to Oracle9iAS 9.0.2.x).

3. The user can deploy to a stand alone version of the OC4J server. Here the user will want to make sure that the stand alone occurrence is using the proper JDK, which will need to be checked and upgraded if necessary.

4. Another option is to use version 9.0.3 of JDeveloper and 9iAS 9.0.3. Make a connection only to ensure that the path and port numbers are correct. After that has been tested, the password is removed. The connection does not need to be used to generate all the files. However, the path and port numbers inside the connection are used by the wizard to generate files which tell the consumers of the services where the services are located. The Oracle rep has given us a set of steps to follow. One downside of this process is that the developer must notify the DBA that there is an ".ear" file that needs to be deployed using the Oracle Enterprise Manager. Only the hostname in that connection is used for the WSDL file generation. The port number can be updated by developer when they generate the WSDL (file location tab). Developer should send the EAR file to the administrator workstation, not the application server machine, the administrator can deploy the EAR file from his workstation.

JDeveloper Modifications

This section covers miscellaneous notes that a developer may need if additional adjustments are to be made to the included code. These notes deal with performing certain actions in JDeveloper that are effected due to the way in which this particular Node is built and deployed.

In using the CSC DNC, we chose to use JDeveloper's built-in support for ANT. That means that you can provide an ANT build file to JDeveloper and say "use this target for building and this one for rebuilding projects". Every time you select a build or rebuild option in JDeveloper, behind the scenes it will run ANT and build the WAR file. This option has a benefit that you can take the project directory and build it anywhere, even on a UNIX machine where there is no JDeveloper installed. Therefore this project does not use an Oracle build script.

Adding your own Libraries to the Node application:

You can add your libraries to the project by copying them into **Node/lib/application** directory. The ANT build will use these libraries to build your application and then place them into the war file in appropriate manner. You also want to add any libraries that you place there, into the JDeveloper project library definition. To do that:

1. Place your library into **Node/lib/application**.
2. Right-click on Node.jpr project and choose Project Settings.
3. Under Development choose libraries.
4. Under Selected Libraries window you will see a library called NodeApplicationLibs click on it.
5. Choose Edit.
6. Next to Classpath text box choose Edit.
7. Click Add Entry and choose library/libraries you want to add (they should be under **Node/lib/application** directory).
8. Click ok.
9. You are done.

Adding your own Classes to the Node application:

ANT Build will pickup and compile all the classes under Node/src directory and jar it into state_ws_impl.jar JAR file. It will then include that file in Node.ear application. If you are getting compilation problems make sure that you have libraries on which this depends placed under **Node\lib\application** as described in Adding Libraries section

New Classes:

Simply choose New option by right-clicking on Sources (in System Navigation Pane).

Existing Classes:

Place your classes into Node/src directory (appropriate hierarchy for packaging must be followed)
Example:
If you want to place a class gov.maine.util.SomeClass you will place SomeClass.java under. Node/src/gov/maine/util/directory.

Exception Handling:

Once we began using the CSC DNC we found that, you have to convert your exceptions into SoapFault in order to throw them properly according to the spec. Additionally you need to create some extra fields inside the AxisFault in order to be compliant with our specification. Throwing a regular exception will cause the toolkit to try to format the AxisFault the best way it can (and may not work correctly).

We have created some helper methods which will help you convert an Exception and create AxisFault out of it according to the specification.

AxisFault fault = FaultHandler.getFault("Client", "Login Failure",
FaultErrorCode.UNKNOWN_USER)
So the first parameter is fault code string, second is fault message, and third is fault error code. There is a list of canned FaultErrorCodes that are compliant with the specification; look at FaultErrorCode's static variables.

Application Server Notes

Note:(additional information on getting 9.0.3 installed):

```
. orasid XXXX (Setting the Oracle Home and other environment settings)
extracted the file to a location:
  */Solaris903/*

./runInstaller
     Component Configuration:
      Selected:
      Oracle Http Server
      Oracle 9iAS Containers for J2EE
      Oracle 9iAS Web Cache
     Infrastructure Summary:
      Single Sign-On Server: <SERVERNAME>.<DOMAIN.NAME>:7777
      Oracle Internet Directory: <SERVERNAME>.<DOMAIN.NAME>:4032
     Oracle Internet Directory:
      Host Name: <SERVERNAME>.<DOMAIN.NAME>
      Port: 4032
      Username: cn=orcladmin (Default)
      Password: ******
     Create Instance Name:
      Instance Name: ias903
      ias_admin Password: *******
     Install....
     run the root.sh script as root...
     Configuration Tools: Running...


     OC4J Config Failed...
     Reading ini file - '<ORACLE_HOME>/j2ee/deploy.ini'


      The following information is available in:
      <ORACLE_HOME>/Apache/Apache/setupinfo.txt
      ----------------------------------------
      Use the following URLs to access Oracle HTTP Server and the Welcome Page:

      http://<SERVERNAME>.<DOMAIN.NAME>:7779
      https://<SERVERNAME>.<DOMAIN.NAME>:4444

      ----------------------------------------
      The Enterprise Manager console can be accessed using the following URL:

      http://<SERVERNAME>.<DOMAIN.NAME>:1810

 ------------------------
 . orasid ias903
 emctl start
 emctl set password manager NEW_PASSWORD

 Download the latest UDDI for 9.0.3...
 Go to the following page http://technet.oracle.com/tech/Webservices/content.html
 Then under downloads choose (Oracle9iAS UDDI Registry 9.03 Upgrade)

 Read PDF file...
      ftp to Application Server
      . orasid iasdb
      cd */uddi9030/ds/uddi/admin
      sqlplus /nolog
      connect /as sysdba
      drop user uddisys cascade;
```

```
      @uddiinst
      sqlldr userid=uddisys/PASSWORD control=naics-97.ctl
      sqlldr userid=uddisys/PASSWORD control=iso3166-99.ctl
      sqlldr userid=uddisys/PASSWORD control=unspsc-73.ctl
      cp the zip file to $ORACLE_HOME
      mv ds ds.old
      unzip orauddi_9030.zip

I went into the OEM console and started all of the services... Next I deployed a new
ear file going through the OC4J_Home:
Step2:
J2EE Application:  C:\working\AppServer\UDDI\orauddi_9030\ds\uddi\j2ee\orauddi.ear
Application Name:  orauddi
Step3:
Url Binding: /uddi
No changes to the rest of the steps...
An error after step five is normal... Ignore

Added the following line to the
      <library path="<ORACLE_HOME>/rdbms/jlib/xsu12.jar"/>
After making this change you need to restart the Web server so the change will take
effect...

Choose the new application: orauddi
Then select datasource & create a new data source
Create Data Source
            Name: UDDI DataSource
            Description: Connection to local database
            Data Source Class: oracle.jdbc.pool.OracleConnectionCacheImpl
            Schema:
            Username:uddisys
            Password:PASSWORD
            JDBC URL:jdbc:oracle:thin:@<SERVERNAME>.<DOMAIN.NAME>:1521:SID
            JDBC Driver:oracle.jdbc.driver.OracleDriver
            Location:jdbc/OracleUddi

            Connection Retry Interval (sec): 1
            Cached Connection Inactivity Timeout(secs): 30
            CREATE...
            Bounce OC4J...

Then you want to put the wsdl file in the htdocs folder:
i.e.: mv *wsdl <ORACLE_HOME>/Apache/Apache/htdocs/.

Restarted all with OEM...

http://<SERVERNAME>.<DOMAIN.NAME>:7779/uddi/
Test the inquiry endpoint... Username: ias_admin/PASSWORD
Then say OK to start your registry...
Then Try all four sample demos

Add the sqlnet.ora file to $ORACLE_HOME/network/admin folder
```

The following can be found on Oracle Metalink:

**Note: The following is more information about not being able to take a commercial 128bit SSL certificate and using the “conversion” tool to import into the Oracle Wallet. The steps under “Oracle Wallet Manger” (pg. 9 steps 1-20) should address the resolution of this bug.

```
Doc ID:  Note:153653.1
Subject:  How do I get 128-bit Strength Encryption with 9iAS or OAS?
Type:  BULLETIN
Status:  PUBLISHED
 Content Type:  TEXT/PLAIN
Creation Date:  01-AUG-2001
Last Revision Date:  08-JAN-2004
```

PURPOSE

Clarification of differences between 40-bit and 128-bit certificates and which certificate is supported on 9iAS and OAS.

SCOPE

The issues are the same for 9iAS as OAS except that OAS does not, and will not, support the so-called 128-bit certificate.

How do I get 128-bit strength encryption with 9iAS or OAS ?
---

4. Some certificate vendors have certificates which they call 40-bit and those they call 128-bit (among other names).

5. No certificates are either 40-bit or 128-bit. Certificates do contain keys but their sizes are numbers like 512, 1024 or 2048 bits and are not used where 40/128 bit encryption keys are used.

6. The 40-bit versions can be used to enable all the common SSL bulk encryption sizes: 40, 56, 64, 112, 128, and 168 bits.

7. The 128-bit versions can ALSO be used to enable all the common SSL bulk encryption sizes: 40, 56, ... 168 bits.

8. The difference between the so called 40-bit and the so called 128-bit certificates are that 128-bit provides:

    a. Faster delivery of the certificate to the customer.

    b. Higher levels of liability coverage (in Dollars)

    c. A "digital right" enabling the "step-up" feature which exists in some of the old, "export" browsers. Browsers which support "step-up" will not perform bulk encryption using key sizes greater than a 40, 56 or 64 bits (depending on when they were made) unless they see a SERVER X.509 certificate containing the special "step-up" digital right. "Step-up" allows them to step-up their bulk encryption key size above 64 bits. When "export" browsers see the "step-up" digital right, they will run at higher bulk encryption sizes generally up to 168 bits.

    Thus, certificates containing the "step-up" digital right are only relevant in situations where people have old "export" style browsers and who are not willing to upgrade to newer browsers which have been available to nearly anyone on Earth since early 2000.

9. The old, Oracle Application Server, 4.0.8.2 and before did not support certificates containing the step-up digital right.

10. X.509 certificates containing the step-up digital right are supported on 9iAS Release 1 Oracle HTTP servers (e.g. 1.0.2.2). Step-up proceeds as described above in these cases.

11. As noted above, use of so-called 40-bit certificates will allow at least 168 bit bulk encryption with any of the 9iAS Oracle HTTP Servers (Releases 1 or 2). 128-bit or 168-bit bulk encryption is allowed whenever people are using

browsers obtained within early 2000 from either Microsoft or Netscape (or any old Domestic Browsers). Note: If using old "export" browsers, the humans operating them can be instructed to acquire newer browsers which both (generally) run faster and will support bulk key sizes up to 168 bits with or without seeing the step-up digital right.

Bottom line: The so called 40-bit certificates allow bulk encryption with up to 168 bit key sizes if used with any Netscape or Microsoft browser obtained since early 2000 or with "domestic" versions of those browsers obtained prior to early 2000. The so called 128-bit certificates will also allow old "export" versions of those browsers to act like newer or "domestic" versions of those browsers for purposes of bulk key size selection. Some organizations may prefer the so called 128-bit certificates when they don't want to mandate that old style "export" browsers be upgraded to versions available to anyone since early 2000 and they want to mandate bulk encryption keys of greater than 64 bits.

Where Oracle 9iAS customers want to mandate 128 bit bulk encryption they can do so with the so called 40-bit certificates if they ensure that their customers use any Netscape or Microsoft browser obtained since early 2000.

Notes when using so-called 40-bit certificates:

a. All browsers obtained from manufacturers since early 2000 can run with 128 bit bulk encryption keys

b. Older export browsers can be allowed to run at less than 128 bit bulk encryption key sizes OR

c. Older "export" browser users can be informed that they should or must upgrade to any of the Netscape or Microsoft browsers made available within the last year.

To ensure the old browsers won't accidentally connect at a lower encryption level. 9iAS has feature to restrict the SSL encryption to be 128 bit by adding following directives to httpd.conf.

<Location the-location-you-want-protected>
SSLRequireSSL
SSLRequire (%{SSL_CIPHER_USEKEYSIZE} >= 128)
</Location>

9iAS 902
========
The so-called Verisign 128 bit SSL Global server Digital Cert is not compatible with the Oracle 9iAS V2 wallet. For Verisign, you must use the so-called Verisign 40 bit SSL certificate.

Note that both the so-called 128 bit and the so-called 40 bit certificates both provide up to 168 bit bulk encryption keys with Oracle 9iAS V2. Note also that no commercial certificate, including either the so-called 128 bit or the so-called 40 bit certificates have less than a 512 bits PKI key (which is the only key in the certificate).

Web Services Security Sample Installation

http://otn.oracle.com/sample_code/tutorials/wspki/wspki_files/Install.html#Get%20a%20Server%20Certificate

SSL Level

By default, the SSLVerifyClient is set to none, it means the SSL only verifies if the SCA of server certificate is valid or not.

In order to do the mutual authentication (which should be in the Node specification), please do the following:

1. Ask administrator to login as iAS administrator.
2. Go to <ORACLE_HOME>/Apache/Apache/conf/httpd.conf
3. Uncomment "SSLVerifyClient"
4. Set it to "require"
5. Restart the apache.
6. Then run your client program again, and it should work also !!!

Export the wallet.
Place the exported wallet on the application server (folder determined by the administrator).