CORBA Southbound Tutorial
Table Of Contents

CSB Tutorial ............................................................................................................. 2
  1.1 Foreword ........................................................................................................... 3
  1.2 Introduction ....................................................................................................... 4
  1.3 Tutorial Tour .................................................................................................... 6
  1.4 Application Overview ...................................................................................... 8
2. TRY IT YOURSELF ............................................................................................... 9
3. APPLICATION DESIGN ...................................................................................... 15
4. MODELING .......................................................................................................... 17
5. IMPLEMENTATION .............................................................................................. 18
  5.1 Project Creation ................................................................................................. 19
  5.2 Resource Modeling ........................................................................................... 21
  5.3 Discovering the ADSL EMS ........................................................................... 23
    5.3.1 Discovery Filter ............................................................................................ 24
  5.4 Map Representation .......................................................................................... 33
    5.4.1 Map Filter ...................................................................................................... 34
  5.5 Fault Handling .................................................................................................. 36
    5.5.1 Status Poll and Notification ....................................................................... 37
  5.6 Performance ....................................................................................................... 40
    5.6.1 Data Decoder ................................................................................................. 41
  5.7 Including the Menu Files .................................................................................. 42
  5.8 Packaging the Project ....................................................................................... 43
6. FAST TRACK IMPLEMENTATION ........................................................................ 44
7. DEPLOYMENT AND TESTING .......................................................................... 45
  7.1 Installation ......................................................................................................... 46
  7.2 Testing ................................................................................................................ 47
8. TROUBLE SHOOTING TIPS ............................................................................. 49
9. KNOWN ISSUES ................................................................................................. 51
10. GLOSSARY ......................................................................................................... 52
11. APPENDIX ......................................................................................................... 53
  11.1 The EMS Simulator .......................................................................................... 53
  11.2 ADSL - An Overview ...................................................................................... 61
12. OTHER TUTORIALS ............................................................................................ 62
CSB Tutorial

CORBA SouthBound

AdventNet Web NMS Tutorial

Version 1.0
1.1 Foreword

AdventNet Web NMS is a protocol neutral framework that serves as a foundation for building Web-based Network Management Systems. AdventNet Web NMS is a highly scalable management infrastructure platform that provides applications combined with tremendous flexibility to customize the framework for specific needs.

Real World Southbound Management Solutions

In Network Management, there is always one or more high level applications that manage a network and its elements. These high level applications obtain the information about the network elements and correlate them for effective network management. In a typical networked environment, a Network Management System monitors the activities of all the devices attached to the Network. At any time, the NMS would require the status and the performance of the network elements. The NMS needs to monitor one or more Element Management Systems irrespective of the protocols used by the EMS.

Some of the Southbound Network Management solutions are

<table>
<thead>
<tr>
<th>Application Management</th>
<th>Managing Enterprise Applications for high availability and performance.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extranet</td>
<td>Managing Supplier, Customer and Partner Extranet application.</td>
</tr>
<tr>
<td>Operational Support System</td>
<td>Managing Service Providers and Equipment Vendors.</td>
</tr>
<tr>
<td>Network Management System</td>
<td>Managing the elements of a wide Network comprising various types of devices.</td>
</tr>
</tbody>
</table>

AdventNet Web NMS

AdventNet Web NMS is a scalable modular framework for rapid development of applications comprehending Product life cycle support and Application deployment architecture. It is a flexible platform that can be easily extended to build feature-rich Management applications.

Application Life Cycle

The steps involved in developing a management application using AdventNet Web NMS are

- Modeling the Managed Elements
- Customizing the NMS Services
- Packaging
- Deployment and Testing
1.2 Introduction

Suppose that you have an Element Management System that manages some ADSL devices. The ADSL network typically consists of ADSL Lines and Channels. To monitor and manage this EMS, you need a Network Management System. This can be accomplished by building an NMS with the help of AdventNet Web NMS. This tutorial depicts how AdventNet Web NMS can be best utilized for effectively managing an EMS using CORBA as the southbound interface. This section is organized as below:

- The Tutorial
- The Intended User
- Prerequisites
- Related Information
- Printed Version
- Tutorial Conventions
- At the end of the tutorial

The Tutorial

The CORBA Southbound tutorial illustrates how AdventNet Web NMS can be used for managing ADSL Element Management Systems with CORBA as the underlying protocol. This is only a sample application which demonstrates how AdventNet Web NMS can be used for your real-time needs. You can develop and customize this application easily with the help of AdventNet Web NMS Studio to meet your requirements.

The Intended User

This tutorial is intended for those who are interested in customizing Web NMS for implementing the CORBA Southbound functionality.

Prerequisites

A good knowledge of Network Management is required for understanding this application. Knowledge of Java is essential.

Related Information

For a detailed Help documentation on AdventNet Web NMS Studio, refer to <Web NMS Home>/Studio Tools/Studio/help/index.html.
For a detailed information on ADSL technology, visit www.dslforum.org and www.itu.int.

Printed Version

To print this tutorial Help Document
1. Install Adobe Acrobat Reader in your machine.
2. Download the PDF version from the URL http://download.adventnet.com/products/webnms/tutorials/corba_southbound_tutorial.pdf
3. Click the Printer Icon.
Tutorial Conventions

<table>
<thead>
<tr>
<th>Font Style</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arial Bold</td>
<td>Filename</td>
</tr>
<tr>
<td>Arial Italic</td>
<td>Directory</td>
</tr>
<tr>
<td>Arial Bold Italic</td>
<td>Methods/Interfaces/Classes</td>
</tr>
<tr>
<td>Courier New</td>
<td>Code Snippet</td>
</tr>
<tr>
<td>Courier New Bold Italic</td>
<td>Highlighting important code snippet</td>
</tr>
</tbody>
</table>

<Web NMS Home> refers to the directory where AdventNet Web NMS is installed. For example, if a Windows user has installed Web NMS in D:\Program Files, then D:\Program Files\AdventNet\WebNMS is <Web NMS Home>.

At the end of this Tutorial, you will be able to

- Manage ADSL Element Management Systems using AdventNet Web NMS.
- Use AdventNet Web NMS Studio.
- Perform Discovery, Status Poll and Data Collection of Network Elements using AdventNet Web NMS.
1.3 Tutorial Tour

This tutorial comprises three stages.

Stage 1

At this stage, you are provided the ready-built tutorial application which can be deployed right away and experienced.

- Try It Yourself - This topic lists the steps to be followed to experience the application.
- Application Design - This topic gives you an overview of the Application Design. It elucidates the various stages involved in building the application.

Stage 2

This stage guides you step by step on how to build the application yourself, including the relevant help on AdventNet Web NMS Studio. Refer to the Implementation section for this stage.

- Creating Project using Studio - Explains the steps to create the Project using Web NMS Studio
- Resource Modeling - Explains how the resources are modeled as Managed Objects
- Discovery - Guides you about creating a Discovery Filter for discovering ADSL EMS
- Map - Guides you about creating a Map Filter to represent the discovered nodes
- Fault - Guides you in handling Fault and Notifications
- Performance - Guides you in performing Data Collection
- Packaging the Project - Guides you through the final step wherein the complete workspace is packaged in to a project
Stage 3

At this stage, you are given the Studio Project of this application. You can compile this project, package and deploy it directly if you are unable to build the project successfully. Further, you can open this project in AdventNet Web NMS Studio and modify it as per your requirement. You can compile, package and deploy it to see the effect of your modifications. Refer to the Fast Track Implementation topic for Stage Three of this tutorial.
1.4 Application Overview

Application Specification

<table>
<thead>
<tr>
<th>Name of the Application</th>
<th>CORBA Southbound Tutorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version of Web NMS</td>
<td>Web NMS Release 4</td>
</tr>
<tr>
<td>Compatibility with other versions</td>
<td>Not compatible with previous versions</td>
</tr>
<tr>
<td>Tools Used and their Versions</td>
<td>AdventNet Web NMS Studio</td>
</tr>
<tr>
<td>Platform Specific Requirements</td>
<td>Studio</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>JDK 1.4</td>
</tr>
</tbody>
</table>

- Application Description
  - Object Modeling
  - Discovery
  - Map
  - Fault Management

Application Description

This tutorial illustrates how AdventNet Web NMS can be effectively used to manage ADSL Element Management Systems. To avoid external dependency of an EMS that manages ADSL lines and channels, we have simulated an EMS. Refer to the EMS Simulator section for more information.

Object Modeling

In AdventNet Web NMS, the information about the Managed Device is modeled for storing in the Database. Here, we have modeled the ADSL lines and Channels as objects.

Discovery

In this tutorial, the Discovery Process of AdventNet Web NMS has been customized to discover the Element Management Systems that manage ADSL devices. The details of the discovered ADSL devices are then stored in the Web NMS Database for proper management.

Map

AdventNet Web NMS provides map creation and map modeling techniques for displaying the discovered network elements and reflecting the status of the Managed Elements. This tutorial enhances the Map module of AdventNet Web NMS to display the ADSL EMS and its constituent lines and channels.

Fault Management

AdventNet Web NMS provides room for effective Fault Management. The Managed Elements automatically send notifications to the server regarding the failure of the nodes. AdventNet Web NMS Fault Management processes the notification and generates corresponding Events and Alerts. In this application, Status Polling is done in frequent intervals and Alarms are generated when an event exceeds a threshold value.
2. Try it Yourself

To experience the application, you can refer to the Deployment and Testing section which lists the procedure for deploying and running the application. You can deploy the NAR that comes along with this tutorial and see its working.

- Before your begin
- Get the ready-built application
- Deploy the ready-built application
- Start the Java IDL Name Server
- Start the EMS Simulator
- Start Web NMS
- View the result

Before your begin

Download a copy of AdventNet Web NMS 4. You need to obtain at least an Evaluation License for the same. Install it. Refer to the Installation Guide of AdventNet Web NMS 4 for help on installation.

Get the ready-built application

The ready-built application is available in the form of a NAR (Nms ARchive) file. Use the NAR bundled with AdventNet Web NMS or download the latest one from the URL http://download.adventnet.com/products/webnms/tutorials/corba_southbound_tutorial.zip.

Extract the zip under the <Web NMS Home> directory.

⚠️ Warning: Take a backup of your configuration files, if you have modified any. Deploying the NAR will overwrite the files. You can restore them after uninstalling the tutorial NAR.

Deploy the ready-built application

Follow the instructions given in the Installation section to deploy the NAR.

Start the Name Server

The Java IDL Name Server should be run for the Simulator to bind its reference. Make sure the path is set to JDK 1.4. Start the Java IDL Name Server by typing the following at the command prompt.

```
tnameserv -ORBInitialPort 1050
```

Start the EMS Simulator

Set the path to JDK 1.4. Execute the `startEMSSimulator.bat/sh` file from the `<Web NMS Home>/tutorials/corba_southbound_tutorial/EMS_Simulator` directory. Click the Bind button.

Start Web NMS

Start the Web NMS Launcher by invoking `WebNMSLauncher.bat/sh` from `<Web NMS Home>` directory.

Double-Click the `Start NMS Server` icon.
View the result

After the server is started, connect to the client by double-clicking the Application Client icon. The Application Client Window is displayed.

Note: Click the images given below to view the enlarged picture.

Map

The ADSL EMS View

Click the ADSLEMS View node in the tree view. The following image shows the EMS Map which displays the EMS managed by the NMS.
Line View

Double click an EMS to display all the ADSL lines managed by that EMS. The following image portrays an ADSL Line View.
Channel View

Double click an ADSL Line to display all the ADSL channels present under that Line. The following image portrays an ADSL Channel View.
Network Events

The notifications generated by Web NMS for the various events that occur in the ADSL EMS are displayed in the Network Events View. These notifications are generated and updated at the time of Status Poll.

Alarms

Network events are propagated as Alarms based on their significance.

Performance

Collected Statistics of an ADSL Line

In the tree view of the application client window, click Performance > Configured Collection.

In the Configured Collection window that appears in the right, select ADSLLine1 from the list of Hosts. Click View > Plot > Collected Statistic. The following image is a snapshot of the line graph of the ADSL Line.
Collected Statistics of an ADSL Channel

In the Configured Collection window that appears in the right, select ADSLChannel1 from the list of Hosts. Click View > Plot > Collected Statistic. The following image is a snapshot of the line graph of the ADSL Channel.
3. Application Design

Aim

To arrive at a design for customizing AdventNet Web NMS so as to manage the ADSL EMS. The requirements with reference to this tutorial application are as below:

- Modeling the ADSL line and its components as Managed resource
- Discovering the ADSL EMS
- Representing the discovered ADSL Lines and Channels in the Map
- Handling Faults
- Data Collection

Modeling the ADSL Line and Its Components as Managed Resources

Requirement

To model the ADSL EMS and its components. Modeling refers to the organized representation of the various attributes of the physical device and its components. These data are stored in the database for persistence.

Task

Identify the properties to be managed. The ADSL Line and the ADSL Channel information need to be modeled as tables in the Web NMS database. The ADSLLine and the ADSLChannel tables store the Line and Channel details respectively. The Line and Channel Configuration Profile are stored in separate tables for easy management. The properties of the ADSL devices that need to be managed are tabulated below:

<table>
<thead>
<tr>
<th>Component</th>
<th>Properties to Be Managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSL Line</td>
<td>Name, Owner Name, Administrative State, Operational State, Customer ID, Line Conditions, Associated Channels, Current Operational Mode, Allowed Operational Mode, Supported Operational Mode, Configuration Profile, and Line Coding.</td>
</tr>
<tr>
<td>ADSL Channel</td>
<td>Name, Owner Name, Line Name, Administrative State, Operational State, and Channel Type.</td>
</tr>
<tr>
<td>ADSL Line Configuration Profile</td>
<td>Name, Owner Name, Rate Mode, Target SNR Margin, Minimum SNR Margin, and Maximum SNR Margin.</td>
</tr>
<tr>
<td>ADSL Channel Configuration Profile</td>
<td>Name, Owner Name, Transmission Rates, and InterLeaveDelay.</td>
</tr>
</tbody>
</table>

Refer to the Modeling section for the class diagram.

The Role of Studio

The objects can be modeled using the Resource Factory of AdventNet Web NMS Studio.

Discovering the ADSL EMS and Its Components

Requirement

To discover the ADSL EMS and then the lines and the channels. The discovered elements need to be added into the Topology database of Web NMS.
Task

Write a Discovery Filter defining the hierarchy of the lines and the channels.

The Role of Studio

The Discovery Filter Wizard of AdventNet Web NMS Studio is used to write the Discovery Filter.

Representing the Discovered Elements in Map

Requirement

The discovered devices need to be displayed in Maps.

Task

The ADSL EMS is laid out in maps with all its subcomponents, namely the ADSL Lines and the ADSL Channels.

The Role of Studio

The Map filter is created by using the Map Filter Wizard of AdventNet Web NMS Studio.

Handling Faults

Requirement

To poll the status of the managed elements at frequent intervals and to create alerts based on the significance of the events.

Task

If an ADSL Line is in faulty state (indicated by Line Condition) for more than a specified period of time (threshold value), an Alarm is propagated, indicating its severity. Some of the line conditions are LossOfSignalQuality, DataInitFailure, ConfigInitFailure and ProtocolInitFailure. The status change is checked by the checkStatus() method of ADSLLine.java.

Data Collection

Requirement

To carry out Data Collection periodically to obtain the performance of the Managed Objects and to represent the collected data in the form of graphs and charts.

Task

Data pertaining to the ADSL Line and ADSL Channel needs to be collected and viewed in the form of Graphs and Bar diagrams.
4. Modeling

This tutorial models the managed objects, namely the ADSL Line and the ADSL Channel as tables in the Web NMS database. The ADSLLine and ADSLChannel tables store the Line and Channel details respectively. The Line and the Channel Configuration Profile are stored in separate tables for easy management. The following class diagram depicts how the information is organized in the Database. ADSLLine is the managed entity that represents an ADSL Line and ADSLChannel is the managed entity that represents an ADSL Channel. Both ATU-C and ATU-R ends of the Channel are represented by this entity.

Class Diagram of CORBA Southbound Tutorial

Refer to the Resource Modeling section for information on how to model the Managed Resource using Web NMS Studio.
5. Implementation

The Studio implementation of CORBA SouthBound Application is explained in this section.

Using AdventNet Web NMS Studio

A Project is created for the Application using the Project Wizard of Studio. Various wizards of Web NMS Studio have been used to customize the services. The OTHER FILES option of Studio is extensively used in this Application.

This section is organized as follows:

- Creating Project using Studio
- Resource Modeling
- Discovery
- Map
- Fault
- Performance
- Packaging the Project

See the following image to know the Studio Architecture. Note that only some of the services represented in the diagram are used by this application.
5.1 Project Creation

To create a Project using AdventNet Web NMS Studio.

Before starting the Studio, make sure that JAVA_HOME is set to j2sdk1.4 in the javaEnv.bat/sh file <Studio Tools Home>.

Invoking the Project Wizard

2. Click File > New > Project to invoke the Project Wizard.

Entering the Project Details

1. Enter the following Project details.
   - Project Name : CSB_User
   - Package Name : com.adventnet.nms.tutorials.csb.adsl
   - Application Name : CSB_User
   - Version : 1.0
2. Click Next.

Entering Device Details

1. Select CORBA from the Protocol Combo box.
2. Specify adsl_ems as Device Type.
3. Click Next

Selecting Service Details

1. Select the following services: -.
   - MANAGEDRESOURCE (selected by default)
   - DISCOVERY
   - MAP
   - PERFORMANCE
   - SECURITY (selected by default)
   - SERVER_CONFIGURATION
   - REBRANDING (selected by default)
2. Click Next.

Entering RDBMS Details

1. Select the database you wish to use.
2. Click Next to proceed.

Entering User Details

1. Click the radio button Single User.
2. Click Next.
Viewing Project Summary

The Project Summary screen appears listing all the properties and services along with the authorized users. To construct the Project Workspace, click Finish.

Once the Workspace is created, the Resource Factory Wizard comes up. Move on to the next section to model your device.
5.2 Resource Modeling

The ADSL Line and its associated components are the resources to be modeled in this tutorial. Here, the ADSL Line has two Channels. The Configuration information about the Line and the Channels are stored in the ADSL Line Configuration Profile Factory and ADSL Channel Configuration Profile Factory respectively. For details, refer to the Creating Managed Resource topic of AdventNet Web NMS Studio Help Documentation.

<table>
<thead>
<tr>
<th>ADSLLine</th>
<th>: This represents the ADSL Line to be managed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSLChannel</td>
<td>: This represents the channel present within the ADSL Line.</td>
</tr>
<tr>
<td>ADSLLineConfigProfile</td>
<td>: This entity contains all the configuration details of the ADSL Lines.</td>
</tr>
<tr>
<td>ADSLChannelConfigProfile</td>
<td>: This represents all the configuration details of the ADSL Channel.</td>
</tr>
</tbody>
</table>

To Model ADSLLine

**Specifying Resource Details**

1. Provide the following information: -.
   - **Name the Managed Resource**: ADSLLine
   - **Select the Parent Resource**: com.adventnet.nms.topodb.ManagedObject

2. Click Next.

**Specifying Attributes of the Managed Resource**

1. Add the following attributes with type String:
   - administrativeState
   - operationalState
   - customerID
   - lineConditions
   - associatedChannels
   - currentOperationalMode
   - allowedOperationalModes
   - supportedOperationalModes
   - configurationProfile
   - ATUCLineCoding
   - ATURLineCoding

2. Click Next.

**Viewing the Source**

1. View the source for Managed Object, Relational Object, Database Schema and Aliases.
2. Click Finish.

Compile ADSLLine created under MANAGEDRESOURCE.

Right click the MANAGED RESOURCE node and select New.

Follow the above set of instructions to model ADSLChannel, ADSLLineConfigProfileFactory and ADSLChannelConfigProfileFactory. The properties to be managed are listed in the following table.
### Component

<table>
<thead>
<tr>
<th>Component</th>
<th>Properties to Be Managed</th>
<th>Type</th>
<th>Parent Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSLChannel</td>
<td>LINENAME, ADMINISTRATIVESTATE, OPERATIONALSTATE, CHANNELTYPE</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATUCRATEMODE, ATURRATEMODE</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>ADSLLineConfigProfile</td>
<td>ATUCTARGETSNRMARGIN, ATURTARGETSNRMARGIN, ATUCMAXSNRMARGIN, ATURMAXSNRMARGIN, ATUCMINSNRMARGIN, ATURMINSTMARGIN</td>
<td>int</td>
<td>com.adventnet.nasa.topodb.ManagedObject</td>
</tr>
<tr>
<td></td>
<td>ATUCFASTMINTXRATE, ATURFASTMINTXRATE, ATUCFASTMAXTXRATE, ATURFASTMAXTXRATE, ATUCILMINTXRATE, ATURILMINTXRATE, ATUCILMAXTXRATE, ATURILMAXTXRATE</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>ADSLChannelConfigProfile</td>
<td>ATUCFASTMAXINTERLEAVEDELAY, ATURFASTMAXINTERLEAVEDELAY, ATUCILMAXINTERLEAVEDELAY, ATURILMAXINTERLEAVEDELAY</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

By default, the properties NAME and OWNERNAME will be present. Add the properties mentioned in the above tabular column in addition to these.

Compile the ADSLChannel, ADSLLineConfigProfile and ADSLChannelConfigProfile nodes created under MANAGEDRESOURCE.
5.3 Discovering the ADSL EMS

Discovery

Discovery module of Web NMS is responsible for discovering elements that reside in the network. It also gathers some basic information about the elements to categorize them based on the information. This tutorial customizes the Web NMS Discovery Process to discover the ADSL EMS and the constituent ADSL lines and channels.

The general sequence of operations of the CORBA Southbound Tutorial Discovery Engine is as follows:

1. Identify the ADSL EMS to be discovered and configure the same in corbaseed.file.
2. Add the discovered EMS in to the topology database.
3. Pass the objects through the Discovery filters.
4. Add the resulting ADSL lines in to the topology database.
5.3.1 Discovery Filter

Let us create a Discovery Filter for this tutorial using the Discovery Filter Wizard of AdventNet Web NMS Studio.

To create a Discovery filter,

1. Start AdventNet Web NMS Studio.
2. Select File > New > Discovery Filter.
3. Specify the class name as ADSLDiscoveryFilter and click OK.

This file will be available as a sub-node under DISCOVERY FILTER in the Studio tree. When you select this file, it opens in JMACS editor. You can make any required changes to the file and compile it.

**Note:** The complete source file is available in <Web NMS Home>/StudioTools/Studio/projects/CSB_Tutorial/Source directory. The code snippets in this topic explain various important portions of the code for easier understanding.

Add the following import statements in the generated code skeleton before importing com.adventnet.nms.topodb.ManagedObject.

```java
import java.util.Hashtable;
import java.util Enumeration;
import java.lang.reflect.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;
import org.xml.sax.*;
```

Now, add the following import statement in between the import statements corresponding to com.adventnet.nms.topodb.ManagedObject and com.adventnet.nms.topodb.TopoAPI.

```java
import com.adventnet.nms.mapdb.MapAPI;
```

Now, add the following lines of code after importing com.adventnet.nms.topodb.TopoAPI.

```java
import com.adventnet.nms.topodb.corba.CORBANode;
import com.adventnet.management.ManagementServer;
import com.adventnet.management.corba.CorbaProperty;
import com.adventnet.management.ManagementServerResultEvent;
```

The following code snippet determines if the discovered node is a CORBA node. If it is not a CORBA node, it returns the Managed Object. Add the code snippet in the filterObject() method.

```java
if (! (managedObject instanceof CORBANode)) return managedObject;
```
Now, add the following lines of code to fetch all the ADSL lines in the discovered EMS.

```java
ADSL.ADSLLineId[] adslLineIdArr = null;
CORBANode emsNode = (CORBANode)managedObject;
ManagementServer ms = ManagementServer.getInstance();
CorbaProperty corbaProp = new CorbaProperty();
DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = fac.newDocumentBuilder();
Document xmlDoc = docBuilder.newDocument();
Element root = xmlDoc.createElement("Attribute");
root.setAttribute("invokeObj", "");
root.setAttribute("operation", "getAllADSLLines");
root.setAttribute("label", "adslLinesRef");
Element rootArr[] = new Element[]{root};
corbaProp.setORBClassName("com.sun.corba.se.internal.iiop.ORB");
corbaProp.setXMLDoc(rootArr);
String ior = emsNode.getStatusPollIOR();
corbaProp.setInvokeObject(ior);
corbaProp.setInterfaceName("EMS.EMSSession");
ManagementServerResultEvent msre = ms.syncSend(corbaProp);
if (msre.getErrString() == null)
{
    Hashtable resHash = (Hashtable)msre.getResult();
    Enumeration e = resHash.elements();
adslLineIdArr = (ADSL.ADSLLineId[])e.nextElement();
}
else
{
    return managedObject;
}

if (first == true)
{
    addEMSViewMap(topoApi);
    first = false;
}

managedObject.setIsContainer(true);
topoApi.addObject(managedObject); // This is for the parentKey which is set for ADSLLine to take effect
addEMSMap(managedObject.getName(), topoApi);
```

Add the following code snippet to obtain the details of the ADSL lines and store them in the database.

```java
int numLines = adslLineIdArr.length;
if(numLines == 0)
{
    System.out.println("******** There are no ADSLLines Managed by this EMS ********");
}
System.out.println("ADSL NODE ");
for(int i = 0 ; i < numLines ; i++)
{
    ADSL.ADSLLineId adslLineId = adslLineIdArr[i];
    ADSL.ADSLLine adslLine = adslLineId.ref;
    CosNaming.NameComponent nc[] = adslLine.getName();
    String lineName = nc[0].id;
    addADSLLineMap(managedObject.getName(), lineName, topoApi);
    ADSL.ADSLLineAll adslLineAll = adslLine.getLineAll(adslLineId);
    ITU_X721.AdministrativeState as = adslLineAll.adminState;
    int adminValue = as.value();
    Field[] fields = as.getClass().getFields();
    String adminState = fields[2*adminValue+1].getName();
    ITU_X721.OperationalState os = adslLineAll.operState;
```
int operValue = os.value();
fields = os.getClass().getFields();
String operState = fields[2*operValue+1].getName();
String customerId = adslLineAll.customerId;
ADSL.LineCondition[] lcArr = adslLineAll.lineStatus;
int numCond = lcArr.length;
String lineConditions = "";
if (numCond > 0)
{
    fields = lcArr[0].getClass().getFields();
    for(int j = 0 ; j < numCond ; j++)
    {
        int lcVal = lcArr[j].value();
        lineConditions = lineConditions+","+fields[2*lcVal+1].getName();
    }
    lineConditions = lineConditions.substring(1);
}
ADSL.OperationalMode com = adslLineAll.currentOperationalMode;
int comValue = com.value();
fields = com.getClass().getFields();
String currOperMode = fields[2*comValue+1].getName();
ADSL.OperationalMode[] aomArr = adslLineAll.allowedOperationalModes;
int numModes = aomArr.length;
String allowedOperModes = "";
if(numModes > 0)
{
    fields = aomArr[0].getClass().getFields();
    for(int j = 0 ; j < numModes ; j++)
    {
        int modeVal = aomArr[j].value();
        allowedOperModes = allowedOperModes+","+fields[2*modeVal+1].getName();
    }
    allowedOperModes = allowedOperModes.substring(1);
}
ADSL.OperationalMode[] somArr =
adslLineAll.supportedOperationalModes;
umModes = somArr.length;
String supportedOperModes = "";
if (numModes > 0)
{
    fields = somArr[0].getClass().getFields();
    for(int j = 0 ; j < numModes ; j++)
    {
        int modeVal = somArr[j].value();
        supportedOperModes = supportedOperModes+","+fields[2*modeVal+1].getName();
    }
    supportedOperModes = supportedOperModes.substring(1);
}
ADSL.LineData atucLineData= adslLineAll.lineAtucData;
ADSL.LineCoding lineCoding = atucLineData.lineCoding;
int lineCodeValue = lineCoding.value();
fields = lineCoding.getClass().getFields();
String atucLineCoding = fields[2*lineCodeValue+1].getName();
ADSL.LineData aturLineData= adslLineAll.lineAturData;
lineCoding = aturLineData.lineCoding;
lineCodeValue = lineCoding.value();
fields = lineCoding.getClass().getFields();
String aturLineCoding = fields[2*lineCodeValue+1].getName();

Now, add the following code snippet to store the ADSL line ID and its reference for Status Polling.
corbaProp = new CorbaProperty();
corbaProp.setOperationType(CorbaProperty.OP_SET);
corbaProp.OBJECT_KEY = lineName;
corbaProp.OBJECT_VALUE = adslLine;
msre = ms.syncSend(corbaProp);
if(msre.getResult() != null)
{
    System.out.println("ADSL Line Object is stored in the Management server for status polling ");
}
corbaProp = new CorbaProperty();
corbaProp.setOperationType(CorbaProperty.OP_SET);
corbaProp.OBJECT_KEY = lineName+"Id";
corbaProp.OBJECT_VALUE = adslLineId;
msre = ms.syncSend(corbaProp);
if(msre.getResult() != null)
{
    System.out.println("ADSL LineId Object is stored in the Management server for status polling ");
}
ADSLLine lineObj = new ADSLLine();
lineObj.setName(lineName);
lineObj.setParentKey(managedObject.getName());
lineObj.setAdministrativeState(adminState);
lineObj.setOperationalState(operState);
lineObj.setCustomerId(customerId);
lineObj.setLineConditions(lineConditions);
lineObj.setAllowedOperationalModes(allowedOperModes);
lineObj.setSupportedOperationalModes(supportedOperModes);
lineObj.setConfigurationProfile("");
lineObj.setAssociatedChannels("");
lineObj.setAtucLineCoding(atucLineCoding);
lineObj.setAturLineCoding(aturLineCoding);
topoApi.addObject(lineObj);

Now, add the following code snippet to obtain the Line Configuration Profile details and to populate the ADSLLineConfProfile table.

//Populate line conf profile table
ADSL.ADSLConfigurationProfileId profileId = adslLineAll.configurationProfile;
ADSL.ADSLConfigurationProfile profile = profileId.ref;
ADSL.ADSLConfigurationProfileAll profileAll = profile.getADSLConfigurationProfileAll(profileId);
String profileName = profileAll.profileName;
try
{
    ADSL.LineConf lineConfAtuc = profileAll.lineConfAtuc;
    ADSL.RateMode rm = lineConfAtuc.rateMode;
    int rmValue = rm.value();
    fields = rm.getClass().getFields();
    String atucTargetRateMode = fields[2*rmValue+1].getName();
    int atucTargetSNRMargin = lineConfAtuc.targetSNRMargin;
    int atucMaxSNRMargin = lineConfAtuc.maxSNRMargin;
    int atucMinSNRMargin = lineConfAtuc.minSNRMargin;
    ADSL.LineConf lineConfAtur = profileAll.lineConfAtur;
    rm = lineConfAtur.rateMode;
    rmValue = rm.value();
    fields = rm.getClass().getFields();
String aturRateMode = fields[2*rmValue+1].getName();
iat urTargetSNRMargin = lineConfAtur.targetSNRMargin;
iat urMaxSNRMargin = lineConfAtur.maxSNRMargin;
iat urMinSNRMargin = lineConfAtur.minSNRMargin;
ADSLLineConfigProfile lineConfProfileObj = new
ADSLLineConfigProfile();
lineConfProfileObj.setName(profileName + "_Line");
lineConfProfileObj.setAtucRateMode(atucRateMode);
lineConfProfileObj.setAtucTargetSNRMargin(atucTargetSNRMargin);
lineConfProfileObj.setAtucMaxSNRMargin(atucMaxSNRMargin);
lineConfProfileObj.setAtucMinSNRMargin(atucMinSNRMargin);
lineConfProfileObj.setAturRateMode(aturRateMode);
lineConfProfileObj.setAturTargetSNRMargin(aturTargetSNRMargin);
lineConfProfileObj.setAturMaxSNRMargin(aturMaxSNRMargin);
lineConfProfileObj.setAturMinSNRMargin(aturMinSNRMargin);
System.out.println("ABOUT TO ADD LINE CONF PROFILE OBJECT
........ ");
topoApi.addObject(lineConfProfileObj);
lineObj.setConfigurationProfile(profileName);
}
} 

Add the following code snippet to obtain the Channel Configuration Profile details and to populate the ADSLChannelConfProfile table.

```java
// Populate channel conf profile table
try {
    ADSL.ChannelConf fastChannelConfAtuc = profileAll.fastChannelConfAtuc;
    int atucFastMinTxRate = fastChannelConfAtuc.minTxRate;
    int atucFastMaxTxRate = fastChannelConfAtuc.maxTxRate;
    int atucFastMaxInterleaveDelay = fastChannelConfAtuc.maxInterleaveDelay;
    ADSL.ChannelConf fastChannelConfAtur = profileAll.fastChannelConfAtur;
    int aturFastMinTxRate = fastChannelConfAtur.minTxRate;
    int aturFastMaxTxRate = fastChannelConfAtur.maxTxRate;
    int aturFastMaxInterleaveDelay = fastChannelConfAtur.maxInterleaveDelay;
    ADSL.ChannelConf interleavedChanConfAtuc = profileAll.interleavedChanConfAtuc;
    int atucILMinTxRate = interleavedChanConfAtuc.minTxRate;
    int atucILMaxTxRate = interleavedChanConfAtuc.maxTxRate;
    int atucILMaxInterleaveDelay = interleavedChanConfAtuc.maxInterleaveDelay;
    ADSL.ChannelConf interleavedChanConfAtur = profileAll.interleavedChanConfAtur;
    int aturILMinTxRate = interleavedChanConfAtur.minTxRate;
    int aturILMaxTxRate = interleavedChanConfAtur.maxTxRate;
    int aturILMaxInterleaveDelay = interleavedChanConfAtur.maxInterleaveDelay;
    ADSLChannelConfigProfile channelConfProfileObj = new ADSLChannelConfigProfile();
    channelConfProfileObj.setName(profileName + "_Channel");
    channelConfProfileObj.setAtucFastMinTxRate(atucFastMinTxRate);
    channelConfProfileObj.setAtucFastMaxTxRate(atucFastMaxTxRate);
    channelConfProfileObj.setAturFastMinTxRate(aturFastMinTxRate);
    channelConfProfileObj.setAturFastMaxTxRate(aturFastMaxTxRate);
```
Add the following code snippet to populate the ADSL Channel table.

```java
ADSL.ADSLChannelId[] associatedChanIds = adslLineAll.associatedChannels;
int numChans = associatedChanIds.length;
String associatedChannels = "";
for(int k = 0 ; k < numChans ; k++)
{
    try
    {
        ADSL.ADSLChannel channel = associatedChanIds[k].ref;
        nc = channel.getName();
        String channelId = nc[0].id;
        associatedChannels = associatedChannels+"","+channelId;
    ADSL.ADSLChannelAll channelAll = channel.getADSLChannelAll(associatedChanIds[k]);

        as = channelAll.adminState;
        adminValue = as.value();
        fields = as.getClass().getFields();
        adminState = fields[2*adminValue+1].getName();
        os = channelAll.operState;
        operValue = os.value();
        fields = os.getClass().getFields();
        operState = fields[2*operValue+1].getName();

        ADSL.ChannelType chanType = channelAll.chanType;
        int chanValue = chanType.value();
        fields = chanType.getClass().getFields();
        String chanTypeStr = fields[2*chanValue+1].getName();

        corbaProp = new CorbaProperty();
        corbaProp.setOperationType(CorbaProperty.OP_SET);
        corbaProp.OBJECT_KEY = channelId;
        corbaProp.OBJECT_VALUE = channel;
        msre = ms.syncSend(corbaProp);
        if(msre.getResult() != null)
```
{ System.out.println("ADSL Channel Object is stored in the Management server for status polling ");
    corbaProp = new CorbaProperty();
    corbaProp.setOperationType(CorbaProperty.OP_SET);
    corbaProp.OBJECT_KEY = channelName+"Id";
    corbaProp.OBJECT_VALUE = associatedChanIds[k];
    msre = ms.syncSend(corbaProp);
    if(msre.getResult() != null)
    { System.out.println("ADSL ChannelId Object is stored in the Management server for status polling ");
    }
    ADSLChannel adslChannel = new ADSLChannel();
    String chanNameWithLineName = lineName+"_"+channelName;
    adslChannel.setName(channelName);
    adslChannel.setAdministrativeState(adminState);
    adslChannel.setOperationalState(operState);
    adslChannel.setChannelType(chanTypeStr);
    adslChannel.setLineName(lineName);
    topoApi.addObject(adslChannel);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
associatedChannels = associatedChannels.substring(1);
lineObj.setAssociatedChannels(associatedChannels);
topoApi.updateObject(lineObj);

Add the following code snippet to set the Callback reference to enable the Notification service between the EMS and the NMS:

CallbackHandler ch = CallbackHandler.getInstance();

    String initHost = ((CORBANode)managedObject).getNameServiceHost();
    String initPort = ((CORBANode)managedObject).getNameServicePort()+"";
    String args[] = {"-ORBInitialHost",initHost,"-ORBInitialPort",initPort};
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    rootPOA.the_POAManager().activate();
    rootPOA.activate_object(ch);
    ITU_X721.Notifications notifObj = ch._this(orb);
    ms = ManagementServer.getInstance();
    corbaProp = new CorbaProperty();
    corbaProp.setOperationType(CorbaProperty.OP_SET);
    corbaProp.OBJECT_KEY = "NMSRef";
    corbaProp.OBJECT_VALUE = notifObj;
    msre = ms.syncSend(corbaProp);
    if(msre.getResult() != null)
    { System.out.println("Web NMS Reference for AdventNet Inc.");
    }
callback is stored in the Management server ");
    }
corbaProp = new CorbaProperty();
corbaProp.setORBClassName("com.sun.corba.se.internal.iiop.ORB");
corbaProp.setXMLDoc(eleArr);
corbaProp.setInvokeObject(ior);
corbaProp.setInterfaceName("EMS.EMSSession");
msre = ms.syncSend(corbaProp);
if (msre.getErrString() == null)
    {
        System.out.println("WebNMS Reference is successfully given to the EMS ");
    }
else
    {
        System.out.println("Cannot give the WebNMS Reference to the EMS ");
    }
corbaProp = new CorbaProperty();
corbaProp.setORBClassName("com.sun.corba.se.internal.iiop.ORB");
corbaProp.setXMLDoc(eleArr);
corbaProp.setInvokeObject(ior);
corbaProp.setInterfaceName("EMS.EMSSession");
msre = ms.syncSend(corbaProp);
if (msre.getErrString() == null)
    {
        System.out.println("WebNMS Reference is successfully given to the EMS ");
    }
else
    {
        System.out.println("Cannot give the WebNMS Reference to the EMS ");
    }

Add the following code snippet which calls the Web NMS MapAPI to modify the Client Window to display the discovered ADSL EMS in the left tree view of the Client Window.

```java
void addEMSViewMap(TopoAPI api)
{
    try
    {
        MapAPI mapapi = com.adventnet.nms.mapdb.MapManager.mapapi;
        java.util.Properties mapprop = new java.util.Properties();
        mapprop.put("mapname", "ADSLEMS_view.netmap");
        mapprop.put("label", "ADSLEMS_View");
        mapprop.put("autoPlacement", "true");
        mapprop.put("anchored", "true");
        java.util.Properties objprop = new java.util.Properties();
        objprop.put("type", "adsl_ems");
        mapapi.addCustomMap("ADSLEMS_view", mapprop,objprop);
    }catch(Exception e)
    {
        e.printStackTrace();
    }
}
```
Add the following code snippet to add the ADSL Line view in the left tree of the Client Window.

```java
void addEMSMap(String emsName, TopoAPI api)
{
    try
    {
        MapAPI mapapi =
com.adventnet.nms.mapdb.MapManager.mapapi;
java.util.Properties mapprop = new
java.util.Properties();
    mapprop.put("mapname", emsName+".netmap");
    mapprop.put("label", emsName+_View");
    mapprop.put("autoPlacement", "true");
    mapprop.put("anchored","true");
    mapprop.put("parentNode","ADSLEMS_view.netmap");
    java.util.Properties objprop = new
java.util.Properties();
    objprop.put("type", "ADSLLine");
    mapapi.addCustomMap(emsName,mapprop,objprop);
    }catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

Add the following code snippet which calls the addCustomMap method of the Map API to add the ADSL channels view in the left tree of the Client Window.

```java
void addADSLLineMap(String parentMapName , String lineName,
TopoAPI api)
{
    try
    {
        MapAPI mapapi =
com.adventnet.nms.mapdb.MapManager.mapapi;
java.util.Properties mapprop = new
java.util.Properties();
    mapprop.put("mapname", lineName+".netmap");
    mapprop.put("label", lineName+"_View");
    mapprop.put("autoPlacement", "true");
    mapprop.put("anchored","true");
    mapprop.put("parentNode",parentMapName+".netmap");
    java.util.Properties objprop = new
java.util.Properties();
    objprop.put("type", "ADSLChannel");
    mapapi.addCustomMap(lineName,mapprop,objprop);
    }catch(Exception e)
    {
        e.printStackTrace();
    }
}
```
5.4 Map Representation

The next step is to lay out the discovered managed objects in map using the Map module of Web NMS. Here, the ADSL EMS, the lines and the channels within the EMS are all laid out in Maps. The hierarchy of display is ADSL EMS(s) > ADSL Line(s) > ADSL Channel(s). The color of the icon representing the line indicates the status of the line. The status change is reflected in the color of the icon. The icons representing the channels vary based on the channel type. All these customizations are carried out in the Web NMS Map module to suit this tutorial.
5.4.1 Map Filter

Let us write a Map Filter to represent the ADSL Lines and Channels in the Web NMS Map.

To create a Map Filter,
1. Start AdventNet Web NMS Studio.
2. Select File > New > Map Filter.
3. Specify the class name as ADSLMapFilter. The Package Name should be com.adventnet.nms.tutorials.csb.adsl and click Next.
4. Click Next in the following Code Builder GUIs until you reach the Source Preview UI. Click Finish.

For details, refer to the Creating Map Filter topic of AdventNet Web NMS Studio Help Documentation.

As Studio presently does not provide extensive support for CORBA nodes, you need to write your own map filter code in the given framework. Edit the Map Filter in Studio and follow the steps given below.

Note: The complete source file is available in <Web NMS Home>/StudioTools/Studio/projects/CSB_Tutorial/Source directory. The code snippets in this topic explain various important portions of the code for easier understanding.

Add the following code snippets in the filterMapSymbols() method.

```java
if ( !(managedObject instanceof ADSLChannel)) return mapSymbols;
```

The above line of code is added to check if the discovered Managed Object is an instance of ADSL Channel.

Add the following code snippet to add the map symbols for the discovered ADSL channels based on their Channel Type.

```java
int numSymbols = mapSymbols.size();
for( int i = 0 ; i < numSymbols; i++)
{
    MapSymbol symbol = (MapSymbol)mapSymbols.elementAt(i);
    ADSLChannel channel = (ADSLChannel)managedObject;
    String channelType = channel.getChannelType();
    if(channelType.equalsIgnoreCase("noChannel"))
    {
        symbol.setIconName("nochannel"+NmsUtil.getImageType());
    }
    else if(channelType.equalsIgnoreCase("fastOnly"))
    {
        symbol.setIconName("fastonly"+NmsUtil.getImageType());
    }
    else if(channelType.equalsIgnoreCase("interleavedOnly"))
    {
        symbol.setIconName("interleavedOnly"+NmsUtil.getImageType());
    }
    else if(channelType.equalsIgnoreCase("fastAndInterleaved"))
    {
```

AdventNet Inc.
symbol.setIconName("fastandinterleaved"+NmsUtil.getImageType());
}
else
if(channelType.equalsIgnoreCase("fastOrInterleaved"))
{
    symbol.setIconName("fastorinterleaved"+NmsUtil.getImageType());
}
}
5.5 Fault Handling

Fault

This tutorial handles some simulated situations such as the change in the ADSL Line Condition. These changes are reflected in the Application Client. Notifications are fired when there is a change in the properties of the line due to a configuration change at the NMS or the Simulator. The NMS reference is used for this purpose. The Line Conditions are returned to the NMS reference during Status Polling. In this tutorial, the status of the ADSL line does not directly depend on the line conditions. The Line Status is changed only when the Line Condition remains the same for a predefined (configurable) threshold level of time. Then an alarm is generated based on the severity of that line.
5.5.1 Status Poll and Notification

Status Polling

To check the status of the line condition, implementation is provided in the \texttt{checkStatus()} method of \texttt{ADSLLine.java}. Status polling is carried out at regular time intervals by calling the \texttt{checkStatus()} method.

Note: The complete source file is available in \texttt{<Web NMS Home>/StudioTools/Studio/projects/CSB_Tutorial/Source} directory. The code snippets in this topic explain various important portions of the code for easier understanding.

The code snippet pertaining to the \texttt{checkStatus()} method is shown below:

```java
public int checkStatus() throws java.rmi.RemoteException
{
    try
    {
        ManagementServer ms = ManagementServer.getInstance();
        CorbaProperty corbaProp = new CorbaProperty();
        DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = fac.newDocumentBuilder();
        Document xmlDoc = docBuilder.newDocument();
        Element root = xmlDoc.createElement("Attribute");
        String name = this.getName();
        root.setAttribute("invokeObj","$OBJ$"+name);
        root.setAttribute("operation","getLineStatus");
        root.setAttribute("label","lineStatus");
        root.setAttribute("returnToUser","true");
        Element arg = xmlDoc.createElement("Argument");
        arg.setAttribute("classType","ADSL.ADSLLineId");
        arg.setAttribute("value","$OBJ$"+name+"Id");
        root.appendChild(arg);
        Element rootArr[] = new Element[]{root};
        corbaProp.setORBClassName("com.sun.corba.se.internal.iiop.ORB");
        corbaProp.setXMLDoc(rootArr);
        ManagementServerResultEvent msre = ms.syncSend(corbaProp);
        ADSL.LineCondition[] lcArr = null;
        if (msre.getErrString() == null)
        {
            java.util.Hashtable resHash = (java.util.Hashtable)msre.getResult();
            /* Since only one value is there in the hash table. */
            java.utilEnumeration e = resHash.elements();
            lcArr = (ADSL.LineCondition[])e.nextElement();
        }
        } catch (Error e) {
            System.err.println("Error in checkStatus() method.");
        }
        int numCond = lcArr.length;
        String lineConditions = "";
        if(numCond > 0)
        {
            java.lang.reflect.Field[] fields = lcArr[0].
            for(int j = 0 ; j < numCond ; j++)
```

AdventNet Inc.
The above code snippet obtains the line condition of an ADSL line and stores it in an array.

**Notification**

Notifications are generated when the Administrative State of a Line or a Channel remains in a faulty state for more than a predefined period of time. The Callback Notifications can be handled in another file. You can create this file using the **OTHERFILES** option of Studio. For details on **Working with Other Files**, refer to AdventNet Web NMS Studio Documentation.

The following method `attributeValueChange()` tracks the change in the attributes of the ADSL Line:

```java
public void attributeValueChange (ITU_X721.AttributeValueChangeInfo attributeValueChangeInfo) {
    try {
        System.out.println("ATTRIBUTE VALUE CHANGE INFO METHOD IS CALLED FROM THE EMS ");
        long time = Long.parseLong(attributeValueChangeInfo.eventTime);
        ITU_X721.MOID moid = attributeValueChangeInfo.source;
        CosNaming.NameComponent[] nc = moid.name;
        String name = nc[0].id;
```
String objectClass = attributeValueChangeInfo.sourceClass;

ITU_X721.AttributeValueChange[] attrValChangeArr = attributeValueChangeInfo.attributeChanges;
ITU_X721.AttributeValueChange attrChange = attrValChangeArr[0];
String attrName = attrChange.attributeName;
String oldVal = attrChange.oldValue.extract_string();
String newVal = attrChange.newValue.extract_string();

String additionalText = attributeValueChangeInfo.additionalText;
// this has the classname of the attribute
System.out.println("ADDITIONAL TEXT -- CLASS NAME
"+additionalText);
Class cls = Class.forName(additionalText);
Field[] fields = cls.getFields();
String oldValue = fields[2*(Integer.parseInt(oldVal))+1].getName();
String newValue = fields[2*(Integer.parseInt(newVal))+1].getName();
TopoAPI tapi = (TopoAPI)NmsUtil.getAPI("TopoAPI");
EventAPI eapi = (EventAPI)NmsUtil.getAPI("EventAPI");
Event e = new Event();
e.setEntity(name);
e.setSeverity(4);
e.setTime(time);
System.out.println("SOURCE CLASS IS (EITHER ADSLLine or ADSLChannel ) "+objectClass);
if(objectClass.equals("ADSLLine"))
{  
e.setText("The value of the "+attrName+" is changed from "+oldValue+" to "+newValue+" in "+name+(ADSLLine) Object ");
ADSLLine line = (ADSLLine)tapi.getByName(name);
line.setAdministrativeState(newValue);
System.out.println("ABOUT TO UPDATE THE LINE OBJECT .... ");
tapi.updateObject(line);
}
if(objectClass.equals("ADSLChannel"))
{  
e.setText("The value of the "+attrName+" is changed from "+oldValue+" to "+newValue+" in "+name+(ADSLChannel) Object ");
ADSLChannel channel = (ADSLChannel)tapi.getByName(name);
channel.setAdministrativeState(newValue);
System.out.println("CHANNEL >.........");
System.out.println("ABOUT TO UPDATE THE CHANNEL OBJECT .... ");
tapi.updateObject(channel);
}
System.out.println("###  ABOUT TO ADD THE Warning Event ........ ");
eapi.addEvent(e);
}catch(Exception e)
{  
e.printStackTrace();
}  
}
5.6 Performance

In this application, data collection is performed periodically and the collected data is passed through the Data Decoder. The Data Decoder is a custom Java code that enables interpretation of polled data in an effective way. Whenever the ADSL Line details are collected, they are passed through the Decoder before getting into the Database. The Decoder checks if the status of the line has changed and if so, updates the database.

Data Collection can be performed for the various attributes of the Line such as the Line Rate, Attainable Rate, and Output Power. The `getLineAtucData` and `getLineAturData` methods are used to obtain the Line details. Data Collection can also be performed for the Channel attributes such as the Channel Rate, Inter Leave Delay, and the Block length. The `getChannelAtucData` and the `getChannelAturData` methods are used to obtain the Channel details.

Collected Statistics

This feature of Web NMS is used to display the collected data in the form of graphs. You can specify the time range and obtain the data as a line graph or a bar graph. Using this feature, you can view the ADSL line data in the form of graphs. Data is retrieved from the database and represented in the form of graphs.
5.6.1 Data Decoder

If the LineCoding of a line is changed, the same is reflected in the ADSLLine table. Using the Other Files option of Web NMS Studio, we have written a Java file ADSLDataDecoder.java. The decode() method performs this task. The following code snippet calls the setAtucLineCoding() (for modification at ATUC end) or the setAturLineCoding() (for modification at the ATUR end) method to update the line coding in the database. For details on Working with Other Files, refer to AdventNet Web NMS Studio Documentation.

```java
if( pdName.equals("LineAtucData") || pdName.equals("LineAturData") )
{
    Hashtable instancesHT = colinfo.getInstanceValuePair(key);
    String instance = instancesHT.remove("lineCodingValue").toString();
    int lineCodeValue = (int)Long.parseLong(instance);
    Class lineCoding = Class.forName("ADSL.LineCoding");
    Field fields[] = lineCoding.getFields();
    String lineCodingVal  = fields[2*lineCodeValue+1].getName();
    TopoAPI topoApi = (TopoAPI)NmsUtil.getAPI("TopoAPI");
    ADSLLine adslLine = (ADSLLine) topoApi.getByName(agent);
    if(pd.getName().equals("LineAtucData"))
    {
        adslLine.setAtucLineCoding(lineCodingVal);
    }
    if(pd.getName().equals("LineAturData"))
    {
        adslLine.setAturLineCoding(lineCodingVal);
    }
    colinfo.setInstanceValuePair(key,instancesHT);
    topoApi.updateObject(adslLine);
}
}
```

```java
catch(Exception e)
{
    e.printStackTrace();
}
return colinfo;
```
5.7 Including the Menu Files

Before packaging the Project as a NAR file, you have to include the required menu files. The files that need to be included are listed below.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
</tr>
</thead>
</table>

Include the mentioned files in the specified location and then proceed with packaging the project.
5.8 Packaging the Project

After creating the project, you need to Save, Compile and Package the same.

Compiling the Project

To compile the project, select the menu Project > Compile.

Packaging the Project

To package the compiled project as a NAR, select Project > Package. This generates the NAR file CSB_User.nar. You can deploy this NAR into Web NMS to install the application. Refer to Installation in the Deployment and Testing section for help on deployment.
6. Fast Track Implementation

If you are not able to complete the Studio Project successfully, use the Studio project that is bundled with AdventNet Web NMS Studio. This project is available in the URL <Web NMS Home>/StudioTools/Studio/Projects/CSB_Tutorial.proj. You can open this project from AdventNet Web NMS Studio and do the permissible changes, save them and compile. Package the application and deploy it to see the changes you have made.
7. Deployment and Testing

This section will guide you through the steps to be followed for deploying and testing the application. Click the relevant topic to know more.

- Installation
- Testing
7.1 Installation

This tutorial is bundled with AdventNet Web NMS 4. You can also download it from the AdventNet's Web Site. Extract the zip under `<Web NMS Home>`.

Steps for Installation

1. Stop the Web NMS Server, if it is running.
2. Set Java Home to JDK 1.3 in `setEnv.bat/sh` found under `<Web NMS Home>`.
3. Run the `DeploymentWizard.bat/sh` file found under `<Web NMS Home>/bin`.
4. Select the NAR Install/Uninstall Tab.
5. Click the Install button to bring up the NAR INSTALLER dialog.
6. Click Browse and select the NAR file `CSB_Tutorial1.0.nar` under `<Web NMS Home>/tutorials/corba_southbound_tutorial/NMS`.
7. Click OK in the NAR INSTALLER dialog to invoke the NmsPwsNarInstaller dialog.
8. Select your Database from the Select Application Database combo box and click Next.
9. Click the Install button to start the installation.
10. Once the installation reaches 100%, click the Close button.
    You will find `CSB_Tutorial1.0` in the list of installed NARs.
11. Click Exit to close the Deployment Wizard.
12. Reinitialize Web NMS by running `reinitialize_nms.bat/sh` from the `<Web NMS Home>/bin` directory.

Note: 1. Java Home must be set to JDK 1.3 in `setEnv.bat/sh` as the `DeploymentWizard.bat/sh` file runs with JDK 1.3 only.
2. `CSB_Tutorial1.0.nar` is the ready-built application that is bundled with AdventNet Web NMS. To deploy the NAR you have created, select `CSB_User` in the NAR Installer Dialog of the Deployment Wizard.

Steps for Uninstallation

1. Stop the Web NMS Server, if it is running.
2. Reinitialize Web NMS by running `reinitialize_nms.bat/sh` from the `<Web NMS Home>/bin` directory.
3. Set Java Home to JDK 1.3 in `setEnv.bat/sh` found under `<Web NMS Home>`.
4. Run the `DeploymentWizard.bat/sh` file found under `<Web NMS Home>/bin`.
5. Select the NAR Install/Uninstall Tab.
6. Select `CSB_Tutorial1.0` from the list of installed NARs and click Uninstall to invoke the NmsPwsNarInstaller dialog.
7. Click the Uninstall button to begin the uninstallation.
8. Click the Close button when the uninstallation reaches 100%.
9. Click Exit to close the Deployment Wizard.

Customization

You can customize this application by modifying the Studio Project (CSB_Tutorial.proj) that is bundled with Web NMS 4. The project `CSB_Tutorial.proj` is present in the `<Web NMS Home>/StudioTools/Studio/Projects` directory. Open the project in AdventNet Web NMS Studio and modify it according to your requirement. Save the Project, Compile and Package it into a NAR. Uninstall the existing NAR and then install the modified one.
7.2 Testing

Starting Java IDL Name Server

Make sure that the Java path is set to JDK 1.4 in setEnv.bat/sh found under <Web NMS Home>. Start the Java IDL Name Server by typing the following at the command prompt.

tnameserv -ORBInitialPort 1050

Starting the EMS Simulator

Make sure that JDK1.4_HOME is set in the startEMSSimulator.bat/sh file. Start the EMS Simulator by running startEMSSimulator.bat/sh present under the <Web NMS Home>/tutorials/corba_southbound_tutorial\EMS_Simulator directory.

Note: If the Name Server is running in another machine, then specify the name of that machine in the ORB Host field. Specify the ORB Port and Name Reference.

Click the Bind button to enable the Simulator tie its reference to the Name Server.

Note: JDKVersionChecker.java (available as an Other File in Studio) checks for the current version of JDK. If it is not JDK 1.4, it returns a message and shuts down the NMS Server.

To start Web NMS Server

1. Set JAVA_HOME to JDK 1.4 in the setEnv.bat/sh file.
2. Execute WebNMSLauncher.bat/sh from <Web NMS Home>. Start the Web NMS Server by clicking the Start Web NMS Server icon.
3. After the server starts, connect to the Application Client by executing the startApplicationClient.bat/sh file from <Web NMS Home>/bin directory.
4. If the Web NMS Server is running in another machine, specify the machine name in the Host field.
5. Specify the User ID as root.
6. Provide the Password as public.
7. Click the Connect button.

The Application client is connected now.

Testing the Application

General Testing

In the client, there will be an ADSL EMS map in which you will see an EMS icon. Double click the EMS icon to view the ADSL Lines managed by this EMS. Double-click an ADSL Line to view the ADSL Channels present under that line. The icon representing a channel depends on the type of the channel. Double clicking the channel icon will display its Managed Object properties.

Status Poll

At the EMS Simulator end,

1. Click ADSLLine1 in the left tree. The properties of the line are displayed in the right view.
2. Select the check box Operational in the Line Condition field.
3. Click the Set Values button.
At the Web NMS Server end,

1. Stop the server, if it is running.
2. Reinitialize the database by running the `reinitialize_nms.bat/sh` file found under `<Web NMS Home>/bin`.
3. Start the Web NMS Server and connect to the client.
4. Click **EMS_View** in the left tree view. Now, you can see that the color of the ADSL Line icon is green.

Similarly, you can test the other conditions using the above procedure by modifying the **Line Condition** field. De-select the **Operational** check box. Now you will find that the adjacent combo box is enabled. Select any one of the conditions listed there and click the **Set Values** button. Start the server after reinitialization. Connect the client. Now, the color of the ADSL Line icon would be orange.
## 8. Trouble Shooting Tips

<table>
<thead>
<tr>
<th>Problem</th>
<th>Reason</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>While starting Web NMS Server, the following exception is thrown: TOPO: Error instantiating filter: com.adventnet.nms.tutorials.csb.adsl.ADSLDiscoveryFilter: java.lang.NoClassDefFoundError: org.omg.PortableServer/Servant</td>
<td>The Server must be running in JDK 1.3. Set it to run in JDK 1.4.</td>
<td></td>
</tr>
<tr>
<td>While starting Web NMS Server, the following error message was displayed: Unable to get the naming service NameService at: org.omg.CORBA.COMM_FAILURE: minor code: 1398079490</td>
<td>The Java IDL Name Server may not be running. Check if it is running.</td>
<td>Start the Java IDL Name Server. Execute the following command: <code>&lt;JDK1.4 Home&gt;/bin/tnameserv -ORBInitialPort 1050</code></td>
</tr>
<tr>
<td>When binding the Name server to Simulator the following exception is thrown: org.omg.CORBA.BAD_PARAM: vmcid: 0x0  minor code:0 completed: No</td>
<td>The Java IDL Name Server may not have started from JDK1.4</td>
<td>Start the Java IDL Name Server. Execute the following command: <code>&lt;JDK1.4 Home&gt;/bin/tnameserv -ORBInitialPort 1050</code></td>
</tr>
<tr>
<td>When you click Bind button in the Simulator UI, the following exception is thrown: org.omg.CosNaming.NamingContextPackage.AlreadyBound: IDL:omg.org/CosNaming/NamingContext/AlreadyBound:1.0</td>
<td>Simulator is already bound to Java IDL Name Server</td>
<td>To bind the Simulator which was already bound with Java IDL Name Server, click the Rebind button and not the Bind button.</td>
</tr>
<tr>
<td>The appropriate entries for ADSLLine Device are made in the corbaseed.file. Even after that the intended device is not getting discovered as ADSLLine.</td>
<td>Entries in the corbaseed.file may not be proper</td>
<td>Ensure that Agent is running. Ensure that the entries that are made in corbaseed.file are complete and proper.</td>
</tr>
<tr>
<td>When two Agents are configured and running, Channels of one of the Device Agent is not getting discovered.</td>
<td>The same Channel Name may be used in the other device also.</td>
<td>Ensure that the appropriate Channel entries are made in <code>EMS_Simulator.xml</code> file. Ensure that the Channel name is not already existing in the other device.</td>
</tr>
<tr>
<td>When the Web NMS Server is started, the entries that were made in the seed.file are not retained.</td>
<td>When the Web NMS Server is started, improper or incomplete entries in the seed.file will be deleted.</td>
<td>Ensure the entries in seed.file are complete and proper.</td>
</tr>
<tr>
<td>Problem</td>
<td>Reason</td>
<td>Solution</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>The ADSL Device Maps exist in the Web NMS Tree, even after un-installing the <strong>CSB_Tutorial1.0.nar</strong>.</td>
<td>-</td>
<td>Ensure that the entries that are made in the seed.file and corbaseed.file after the NAR installation are removed after un-installing the NAR.</td>
</tr>
<tr>
<td>After un-installing the <strong>CSB_Tutorial1.0.nar</strong>, the entries in the seed.file and corbaseed.file are not restored with pre-installation values.</td>
<td>-</td>
<td>Manually restore the entries in the seed.file and corbaseed.file to the pre-installation values.</td>
</tr>
</tbody>
</table>
9. Known Issues

- This tutorial works with JDK 1.4 only. The Web NMS Server, the Java IDL Name Server and the EMS Simulator must be running in JDK 1.4. Note that after deploying the tutorial NAR, you need to set the Java Home to JDK 1.4 in `setEnv.bat/sh` since the Deployment Wizard tool of Web NMS works with JDK 1.3 only.
- There is a Map View associated with 'Corba Nodes' in the Web NMS Application Client which will be added when a CORBA node is discovered. However, the map pertaining to this tutorial is available in the node ‘ADSLEMS_View’.
10. Glossary

ITU-T - International Telecommunications Union - Telecommunication Standardization Sector is the primary international body for fostering cooperative standards for telecommunications equipment and systems.

ADSL - Asymmetric Digital Subscriber Line is a technology for transmitting digital information at a high bandwidth on existing phone lines to homes and businesses. ADSL is asymmetric in the sense that it uses most of the channel to transmit downstream to the user and only a small part to receive information from the user.

ATU-C - ADSL Transceiver Unit at the Central Office end.

ATU-R - ADSL Transceiver Unit at the Customer Premise end.

IDL - Interface Definition Language is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.

POTS - Plain Old Telephone System.

QAM - Quadrature Amplitude Modulation is a modulation scheme in which two sinusoidal carriers are used to transmit data over a given physical channel. QAM and its derivatives are used in both mobile radio and satellite communication systems.

CAP - Carrier-less Amplitude/Phase modulation describes a version of QAM in which incoming data modulates a single carrier that is then transmitted down a telephone line. The carrier itself is suppressed before transmission (it contains no information and can be reconstructed at the receiver) and hence the adjective carrier-less.

DMT - Discrete Multi-Tone describes a version of multicarrier modulation in which incoming data is collected and then distributed over a large number of small individual carriers, each of which uses a form of QAM modulation. DMT creates these channels using a digital technique called Discrete Fast-Fourier Transform. DMT is the basis of ANSI Standard T1.413.
11. Appendix

11.1 The EMS Simulator

- About the Simulator
- ADSL.idl
- ITU_X721.idl
- ITU_X721Const.idl
- Naming.idl
- EMSSession.idl
- Line Properties
- Line Data
- Channel Properties
- Channel Data
- Configuration Profiles

About the Simulator

This add-on application simulates an Element Management System that manages ADSL Lines and Channels. Every CORBA agent running on a device implements an IDL. To manage the lines and the channels, this tutorial implements the following five IDLs: -.

ADSL.idl

This IDL defines the interface between AdventNet Web NMS Client and the EMS Simulator. This IDL defines the entities that manage the ADSL transport. The entities are the ADSL Line, ADSL Channel and the ADSL Configuration Profile.

ITU_X721.idl

This IDL defines the Managed Object Interface which is the parent interface from which all the other interfaces inherit. This IDL also defines the Notification interface.

The following image illustrates the implementation of multiple interfaces from the ADSL Managed Object interface.

ITU_X721Const.idl

This IDL defines constants for some of the general attributes of a Managed Entity such as ProbableCause and SecurityAlarmCause.
Naming.idl

This IDL provides the Java IDL Naming Service which is the interface between the NMS and the EMS.

EMSSession.idl

This IDL plays the key role in the data transfer between the NMS and the EMS. This is responsible for the discovery of the EMS that is configured in `<Web NMS Home>/conf/corbaseed.file`. The EMS discovery is followed by the discovery of ADSL Lines and Channels.

The sequence of operations performed by EMSSession.idl is as follows.

i. Obtain the NMS reference by invoking the `getNMSRef()` method which takes the NMS Object as the argument.

ii. Discover the EMS configured in `corbaseed.file`.

iii. After discovering the EMS, look for the ADSL Lines within the EMS by calling the `getAllADSLLines()` method.

The Simulator

The EMS simulator has been built using AdventNet Web NMS Bean Builder. The simulator that is bundled along with this tutorial manages one ADSL Line which has two ADSL Channels. You can customize this EMS to manage more lines and channels. The details of the ADSL lines and channels can be configured in the `EMS_Simulator.xml` file present in the `<Web NMS Home>/tutorials/corba_southbound_tutorial/EMS_Simulator/conf` directory.
**Line Properties**

Click the node **ADSLLine1** in the left tree view. The properties of **ADSLLine1** are displayed in the right view. The following image shows a typical ADSL Line view of the EMS Simulator. You can modify any of the enabled properties and click the **Set Values** button for your changes to take effect. The properties of the line can also be modified by editing the corresponding parameters of the **ADSLLINE** tag in **EMS_Simulator.xml**.
Line Data

Click ATUC or ATUR found under LINE DATA to view or modify the details of the ADSL Line at the ATUC and ATUR end. You can modify the Line Coding and other properties and click the Set Values button for your changes to take effect. The following image shows a snapshot of the configuration details of a line at ATUC end. The properties of the Line Data can also be modified by editing the corresponding parameters of the LINEDATA tag present in EMS_Simulator.xml.

ADSL Line Data at the ATUC end
Channel Properties

The Administration State, the Operation State and the Type of channel can be viewed and modified by clicking the ADSL Channel name under ADSL CHANNELS. In the following image ADSLChannel1 and ADSLChannel2 are the names of the ADSL Channels. You can also modify these values by editing the corresponding parameters of the `ADSLCHANNELS` tag present in `EMS_Simulator.xml`.

![Properties of the ADSL Channel](image-url)
Channel Data

The details of a channel at the ends ATUC and ATUR can also be viewed and modified by clicking the respective node in the left tree view. The following image shows the channel data at the ATUC end of the channel ADSLChannel1.

ADSL Channel Data at the ATUC End

You can also modify these values by editing the corresponding parameters of the CHANNELDATA tag present in EMS_Simulator.xml.
Configuration Profiles

The configuration profiles of the ADSL Line and the Channel are present under ADSL CONFIG PROFILE. Click the LINE CONF tree item to view the configuration of the ADSL Line at the ATUC and ATUR ends.
Similarly, the channel configuration details at both the ends can be obtained by clicking the CHANNEL CONF tree item.

**Channel Configuration Details**

**Note:**
1) The changes made to `EMS_Simulator.xml` will take effect only when the EMS Simulator is started the next time.
2) Only the properties that are used for Data collection and Notification of NMS are editable in the UI. However, you can configure all the properties by editing the `EMS_Simulator.xml` file.
11.2 ADSL - An Overview

Asymmetric Digital Subscriber Line (ADSL) is a modem technology that converts existing copper telephone lines into access paths for multimedia and high-speed data communications, and maintains the regular telephone voice services. This technology is designed to solve the most severe bottleneck in the data access network between the Central Office and the customer, or end-user. ADSL is one of the xDSL standards and products designed to provide high data rate communication over the final 'copper loop' of the PSTN network. ADSL transmits two separate data streams with more bandwidth devoted to the downstream than to the upstream. ADSL takes advantage of the fact that most of its applications such as Internet, video conferencing, interactive multimedia, and Video-on-Demand function perfectly with a relatively low upstream data rate.

ADSL System

The ADSL operates over a single, unconditioned twisted copper pair of wires. Its connection is via modem pairs, one at the user end and the second at the Central office. ADSL service is operated in conjunction with existing the Plain Old Telephone Service (POTS). Telephony signals and ADSL signals are separated and combined by the POTS splitters, one at the Central Office and one at the ADSL service user, or subscriber’s home.

The following figure shows the overview of ADSL Line.
12. Other Tutorials

AdventNet Web NMS is vast in its capability to serve its different class of users. It would be hard for anyone to understand all its features at one time. We strongly recommend you to go through some of our other tutorials to get a feel of what could be done on our Web NMS.

Building an Element Management System

This tutorial provides illustrative examples to guide the developer through designing an EMS. Design aspects and the usage of Web NMS tools to simplify the development of an EMS are elaborated here.

CORBA Northbound Tutorial

This CORBA Northbound Tutorial shows how easily AdventNet Web Network Management System (NMS) can be used for distributed NMS applications by providing CORBA northbound support to interact with other Northbound Operational Support Systems (OSS).

Managing a TL1 Device

This tutorial guides the developer on how to manage a device using the TL1 Protocol. Various features in AdventNet Web NMS to manage a TL1 Device are explained. The design aspects and the usage of Web NMS tools to simplify the development of an EMS and the ease of managing a device using the TL1 Protocol are described here.

XML Southbound Tutorial

This tutorial builds an EMS with the information got from an XML-enabled device. From the responses received, the EMS chassis is built and some other functions are included.