# Table Of Contents

**CORBA NORTHBOUND TUTORIAL** ................................................................. 2  
1.1 Foreword .......................................................................................... 3  
1.2 Introduction ...................................................................................... 5  
1.3 CORBA Northbound Tutorial Tour .................................................... 7  
1.4 Application Overview ...................................................................... 8  

**2. TRY IT YOURSELF** ............................................................................ 11  

**3. APPLICATION DESIGN** .................................................................. 19  

**4. IMPLEMENTATION - WEB NMS** ..................................................... 22  
4.1 Creating CNB Project ....................................................................... 23  
4.2 Configuring CORBA Parameters ....................................................... 25  
4.3 Modeling CORBA Object .................................................................. 26  
4.3.1 ChannelData ................................................................................ 27  
4.4 Implementing CORBA Support in Web NMS ...................................... 28  
4.4.1 CorbaServer ................................................................................ 29  
4.4.2 Server Configuration ................................................................... 31  
4.5 Notifying Web NMS Alerts to OSS .................................................... 32  
4.5.1 OSS .............................................................................................. 33  
4.6 Packaging the Project ...................................................................... 36  

**5. COMMON IMPLEMENTATION** ........................................................ 37  
5.1 Creating Custom IDL for OSS-NMS Data Transaction ..................... 38  
5.1.1 OSS.idl ........................................................................................ 39  

**6. OSS IMPLEMENTATION** ................................................................. 40  
6.1 Opening the Bean Builder Project ...................................................... 42  
6.2 Creating Operational Support System: OSS_Server .......................... 43  
6.3 CORBA Interface for OSS: OSSImpl .............................................. 45  
6.4 Compiling the Project ...................................................................... 47  

**7. FAST TRACK IMPLEMENTATION** ...................................................... 48  

**8. DEPLOYMENT AND TESTING** ......................................................... 49  
8.1 Installation Notes .............................................................................. 49  
8.2 Testing ............................................................................................... 51  
8.3 Channel Data Simulation .................................................................. 57  

**9. KNOWN ISSUES** .............................................................................. 58  

**10. TROUBLE SHOOTING TIPS** .......................................................... 59  

**11. GLOSSARY** ................................................................................. 62  

**12. OTHER TUTORIALS** ...................................................................... 63  

*AdventNet Inc.*
This Tutorial shows how AdventNet Web NMS can be used for distributed NMS applications by providing CORBA Northbound support to interact with Operational Support Systems (OSS).
1.1 Foreword

AdventNet Web NMS aims at providing real world network management solutions to telecom and enterprise markets. It meets the demand of the market for advanced network management features. It addresses the market's need of the hour i.e., shortest possible deployment time.

CORBA Northbound tutorial will demonstrate how the above market expectations are met by AdventNet Web NMS.

- Real World Northbound Management Solutions
- Why AdventNet Web NMS?

Real World Northbound Management Solutions

A Network Manager, sitting in a remote location would like to review the performance of his/her distributed network and would like to have the performance data at a central place gathered from various NMS, independent of communication protocols involved. AdventNet Web NMS provides an easy solution to all such requirements.

The major requirements for Network Management Systems are to interact with each other and be managed by another higher level manager. In any typical distributed application, various Network Management Systems will interact with one another and exchange data related to Faults, Configuration and Performance using CORBA, RMI, HTTP, SNMP for communicating with one another. An OSS, which may manage these NMSs, would like to take quick decisions and reconfigure networks if and when faults occur, to maintain high availability of the network resources.

Real world northbound management applications, which can be made available on AdventNet Web NMS, categorized by specific domains are as follows:

**Provisioning and OSS** : End-to-end XML template driven service provisioning for service providers and equipment vendors.

**Application Management** : Managing Enterprise applications for high availability and performance.

**Extranets** : Managing Supplier, Customer, and Partner Extranet applications.

**Managed Services on Subscription** : Outsourcing Network and Applications management by Enterprises.

The list of Web NMS applications goes on.

Why AdventNet Web NMS?

AdventNet Web NMS fulfills your specific network management needs. It comes with the following most sought after features in the market:

- Massive scalability
- High availability
- Customization
  - Modeling managed systems
  - Extending management services
  - Supporting variety of management protocols
  - Various deployment options

It can be customized and extended to suit your needs. The extensibility makes the design of the application more organized.
The customization addresses the specific needs of the application to manage your custom equipment. AdventNet Web NMS Studio is a developer suite with rich tools. It is bundled in the AdventNet Web NMS product itself. This suite reduces the development life cycle time to bare minimal. This in turn brings a host of benefits:

- The time taken to deploy the application is lesser compared to the conventional development and deployment techniques.
- The human resources requirement is reduced to a fraction of what is required for conventional techniques.
- The tool supports user from the level of novice to professional. The tool contains UI based Wizards to accomplish all the simple tasks if you are a novice user and makes room for custom code if you are a professional to handle advanced tasks.
- The required skill level of the user is also brought down. For example, you do not require Java knowledge to use the tool and only network management and your element domain knowledge will suffice.

All these benefits put together will make AdventNet Web NMS a wise choice for your network management solution.

Use of AdventNet Web NMS Studio will make the development of the application virtually very little or zero coding. This makes the development faster.
1.2 Introduction

The purpose of this tutorial is to guide you through designing a CORBA Northbound application and to provide you with working and illustrative examples to help you understand the choices to be made during development. This also explains how to build CORBA Northbound application using AdventNet Web NMS and the AdventNet Web NMS Studio.

The Tutorial

Consider a real-life scenario wherein you want to expose the AdventNet Web NMS Resources to OSS (Operational Support System) and you want to carry out operations on Web NMS objects from OSS. Then, this tutorial will help you to achieve the same by building a CORBA Northbound application, using AdventNet Web NMS.

This tutorial explains

- How the AdventNet Web NMS resources are exposed to OSS, so that they can be used for management?
- How to bind the NMS and the OSS using the IDLs provided by AdventNet Web NMS, to establish connection and transact data between them?
- How to achieve the Device Management functions from OSS through AdventNet Web NMS?
  - How to get the status of a device from OSS through AdventNet Web NMS?
  - How to make the OSS to get the Alert information from AdventNet Web NMS, using CORBA callback mechanism?
  - How to configure a device from OSS through AdventNet Web NMS, by making CORBA calls?

This tutorial helps you to build only a sample application. In which, you will be creating Custom process in Web NMS to carry out OSS operations locally in Web NMS and configuring corba_parameters.conf file to enable CORBA in Web NMS and to expose AdventNet Web NMS resources to OSS through CORBA.

This limited scope example application will serve only to demonstrate what can be built on AdventNet Web NMS and it is only a subset of the capabilities of AdventNet Web NMS.

This topic covers the following details of the tutorial:

- The Intended User
- Prerequisites
- Related information
- Printed version
- Tutorial conventions
- At the end of the tutorial

The Intended User

This tutorial is intended for those, who are interested in developing a Northbound application with CORBA technology, using the AdventNet Web NMS.

Prerequisites

To develop this tutorial application, you must have a good knowledge of Network Management System, CORBA, and Northbound concepts.

Knowledge of Java programming will be an added advantage.
Related Information

This tutorial provides concise information about AdventNet Web NMS Studio and AdventNet Web NMS. For detailed information about these, refer to the Web sites listed below:

1. AdventNet Web NMS Studio documentation - from the following URL: 
   <Web NMS Home>\StudioTools\Studio\help\index.html

2. AdventNet Web NMS documentation - from the following URL: 

Printed Version

To print this tutorial, follow these steps:

1. Ensure that Adobe Acrobat Reader is installed on your system.
2. Download the PDF version of this document from the following URL: 
3. Click the printer icon in Adobe Acrobat Reader.

Tutorial Conventions

The following table lists the typographic conventions followed in this tutorial:

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

<Web NMS Home> - refers to the AdventNet Web NMS root directory
<Java Home> - refers to the JDK root directory

At the End of the Tutorial

From this tutorial, you will learn how to:

- Build a Northbound application using AdventNet Web NMS.
- Expose AdventNet Web NMS resources to OSS.
- Achieve Fault, Configuration, Performance, and Security management operations on Web NMS objects from OSS.
- Use the AdventNet Web NMS Studio.
1.3 CORBA Northbound Tutorial Tour

Welcome to the CNB Tutorial Tour
This tutorial is in three tiers.

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.

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.

Hope you will enjoy this tour and experience easy and quick Application development.
1.4 Application Overview

This section provides an overview of the Tutorial Application on how to develop a Northbound application for AdventNet Web NMS to interact with OSS.

AdventNet Web NMS offers the necessary APIs to provide multiprotocol Northbound support. For CORBA, it provides the IDLs needed by OSS for getting Event and Alert information. It also provides the IDLs for gathering the Topology, Map, and Poll data. Custom IDLs can be written to extend the basic functionality provided by AdventNet Web NMS.

- Application Specification
- Implementation in Nutshell

Application Specification

**Name of the Application**: CORBA Northbound Tutorial  
**Version of Web NMS used**: Web NMS Release 4  
**Version of JDK**: JDK 1.3  
**Compatibility with other Versions**: Not compatible with previous versions of Web NMS  
**Tools used and their Versions**: AdventNet Web NMS Studio 4.0  
**Platform specific requirements**: No special requirements  
**Requirements to run the Application**: No special requirements

Implementation in Nutshell

The tutorial application covers various aspects of Northbound management. The implementation is in two parts.

- Part one is NMS implementation.
- Part two is OSS implementation.

Various modules of AdventNet Web NMS are used to achieve the Northbound management objectives as listed below:

---

**NMS implementation**

- How to extend the Web NMS Managed Object as CORBA Managed Object?
- How to expose Web NMS resources to OSS?
- How to implement CORBA support in Web NMS?
- How to notify Web NMS Alerts to OSS?

**OSS implementation**

- How to implement CORBA support in OSS?
- How to access the CORBA objects through Web NMS?
- How to build the Client User Interface?

**Common to both**

- How to transact data between Web NMS and OSS using CORBA protocol?
NMS Implementation

How to Extend the Web NMS Managed Object as CORBA Managed Object?

You can extend the core Web NMS object (i.e., Managed Object) to CORBA Managed Object, by writing a Java class file named ChannelData.java. It implements a CORBA interface called ChannelData_CIOperations.

How to Expose Web NMS Resources to OSS?

You can expose all the Web NMS resources to OSS by exposing the Web NMS APIs. This is achieved by configuring the corba_parameters.conf file. AdventNet Web NMS APIs are accessible via CORBA.IDLs.

How to Implement CORBA Support in Web NMS?

You can write a new CORBA process and configure the same in Web NMS to implement CORBA support in Web NMS.

The custom CORBA process will initialize the ORB, create and connect the servant OSS to ORB, resolve the naming context, and bind the Servant Object Reference in Name Service. This is achieved by writing a Java class file named CorbaServer.java. You need to configure this class in the NmsProcessesBE.conf file.

How to Notify Web NMS Alerts to OSS?

You can get Web NMS Alerts notified to OSS. This is achieved by writing a Java class file named OSS.java. This class registers this process as an Alert observer in Web NMS. This receives the Alert notifications from Web NMS and by using the CORBA Callback mechanism transmits the Web NMS Alerts to OSS.

OSS Implementation

How to Implement CORBA Support in OSS?

You can implement CORBA support in OSS by writing a Java class named OSS_Server.java. This class will initialize the ORB, create and connect the servant OSS to ORB, resolve the naming context, and bind the Servant Object Reference in Name Service.

How the OSS Gets Access to CORBA Objects (i.e., Web NMS Resources)?

You can make the OSS to access the Web NMS resources by writing a Java class named OSSImpl.java. This class is an API interface for accessing and updating the North bound CORBA Objects in the device through Web NMS.

How to Build the Client User Interface?

You will be able to develop Client User Interface using AdventNet Bean Builder.
Common to Both

How to Transact Data Between Web NMS and OSS Using CORBA Protocol?

You will have to write a custom IDL called OSS.idl to achieve the above mentioned objective. The OSS.idl uses the IDLs provided AdventNet Web NMS. They are Common.idl, Topology.idl, and Alert.idl. Once you have the OSS.idl, the next step is to generate the Java code for the OSS.idl using idltojava compiler. This will generate necessary Java files from the IDL, to be used with both Web NMS and OSS.
2. Try It Yourself

This section lists out the steps to deploy the ready built application. This topic will help you run the application and view the results. Before going into the details of building the application, you can experience what the application will exactly deliver.

To execute this application, follow the steps given below:

- Before You Begin
- Get the Application’s NAR File
- Deploy the NAR in AdventNet Web NMS
- Run the Application
- View the Result

Before You Begin

2. For trying this tutorial, you have to take a Trial User License from AdventNet, Inc.
3. Get it installed in your machine.

For pertinent information, refer to the following document resources in the Installation Guide of AdventNet Web NMS 4:

- System Requirements
- Startup Options

Get the Application’s NAR File

The working example application comes bundled with AdventNet Web NMS, as a NAR file. Alternatively, you can download the latest version of the tutorial from the AdventNet Web site and use the NAR in it.

a. Use the bundled NAR available in the following path:

<Web NMS Home>/tutorials/corba_northbound_tutorial/NMS/CNB_Tutorial1.0.nar

OR

b. Download the latest version from the Web site at the following URL:

http://download.adventnet.com/products/webnms/tutorials/corba_northbound_tutorial.zip

Unzip the zip file in the <Web NMS HOME> directory.

Select the NAR file mentioned below from the <Web NMS Home>/tutorials/corba_northbound_tutorial/NMS directory.

CNB_Tutorial1.0.nar

Deploy the NAR in AdventNet Web NMS

Carryout the instructions given in the Installing the application section to deploy the NAR file.
Run the Application

- **Start Transient Naming Server (TNameServer)**
  
  Start the tnameserver by entering the following command in the command prompt (Make sure that the JDK path is set in that command prompt).
  
  \texttt{tnameserv -ORBInitialPort 1050}

- **Start NMS**
  
  1. Start the Web NMS Launcher by invoking \texttt{WebNMSLauncher.bat/sh} file in the \texttt{<Web NMS HOME>} directory.
  2. Click \texttt{Start NMS Server} icon in the \texttt{Web NMS Launcher}.

  [OR]

  Run \texttt{startnms.bat/sh} from \texttt{<Web NMS Home>/bin} directory

- **Start OSS Client**
  
  Start the OSS Client, by invoking \texttt{run_oss_client.bat/sh} file in the \texttt{<Web NMS HOME>/tutorials/corba_northbound_tutorial/oss/bin} directory.

  \begin{center}
  \begin{tabular}{|l|}
  \hline
  \textbf{Note:} After installing the NAR, ensure that EnableCORBA is set to \texttt{true} in the \texttt{corba_parameters.conf} file in the \texttt{<Web NMS Conf>/conf} directory. Otherwise, the Client will not be getting connected. \\
  \hline
  \end{tabular}
  \end{center}

View the Result

The OSS client you see is a prototype sample OSS and has been built using the Bean Builder of AdventNet Web NMS. The OSS Client opens up with the \textbf{MO Panel} screen.

The screen shot given below shows how the OSS Client UI will look like.
OSS Client - MO Panel

Select the Alarms panel in tree. You will see the Web NMS alerts exposed through CORBA.
Select the Events Panel in tree. You will see the Web NMS events exposed through CORBA.
Events Panel

This way, Web NMS resources are exposed to OSS.

In the MO Panel, select a Managed Object by double-clicking on it. You will see the **Status Screen** with **MO Status** tab displayed and it has other two tabs viz., **Alert Messages** and **Channel Messages**.

In the MO Status tab, you will be able to view the details of selected managed object of Web NMS.
MO Status Update
In the Alert Messages tab, you will be able to view all the Alert messages of Web NMS.

Alert Notifications through callback
In the Channel Messages tab, you will be able to view the details of the Channel information of the selected MO.

Channel Related Messages.

In the MO Status tab, apart from the status of MO, you have the facility to configure Network devices. In the UI, you can find Device Status combo box containing all possible status values that can be set in the Network devices and Set button to configure the status of the Network device. The UI also displays both the MO Status before the Status update and after the Status update. This way, the OOS configures the Network devices through AdventNet Web NMS.
3. Application Design

This topic explains the design of the CORBA Northbound application.

Aim

- To come up with the design on customizing the AdventNet Web NMS platform features in order to provide CORBA Northbound.
- To design a sample OSS Client User Interface.
- To design the IDL common to both OSS and NMS.

Northbound Management Requirements

NMS

- Modeling the Channel Data of the Switch device as Managed Resource
- Enabling and Configuring CORBA Service in NMS
- Configuring CORBA as Web NMS module
- Creating CORBA Process
- Managing the Alerts of NMS from OSS

OSS

- Creating OSS Server
- Creating CORBA Interface for OSS

Common

- Creating the IDL common to both NMS and OSS

NMS

Managed Resource Modeling

Objective

Modeling a device enables you to represent the various attributes and the behavior of the corresponding physical device and its components in a convenient way so as to reflect their current state at any point of time. The Northbound NMS stores these persistent data in the database.

Tasks

Define the resources to be managed by the OSS through Northbound NMS. The properties of the Device Components that will be used for modeling the resources are as given below.

| ChannelData | currentChannelRate, previousChannelRate, interleavedDelay, crcBlockLength |

The Role of Studio

Using the Other Files of the Studio, the above resource can be modeled.
Note: Using the Resource Factory to model object will create a Managed Object, RelationalMO, schema, alias etc. But no table is maintained for this at present. Hence, the Other Files option of Studio is used.

Enabling and Configuring CORBA Service in NMS

Objective
To enable CORBA protocol and configure the CORBA Parameters in the Web NMS.

Tasks
- Enable CORBA protocol and configure the CORBA Parameters in the corba_parameters.conf file.

The Role of Studio
Studio does not support this file. You have to configure the above file separately.

Configuring CORBA as Web NMS module

Objective
To configuring CORBA as Web NMS module.

Tasks
- Configuring CORBA as Web NMS module in the NMSProcessesBE.conf file.

The Role of Studio
In the Studio, the Server Configuration can be used to configure CORBA as Web NMS module.

Creating CORBA process

Objective
To create CORBA Process, in order to handle the custom created OSS Client.

Tasks
- Create the CORBA process by writing a class CorbaServer.

The Role of Studio
Using the Other Files of the Studio, the above class can be created.

Managing the Alerts of NMS from OSS

Objective
To monitor and manage the Alerts in the NMS by the OSS.

Tasks
- Convey the Alerts from NMS to OSS by writing a class OSS.
The Role of Studio

Using the Other Files of the Studio, the above class can be created.

OSS

Creating OSS Server

Objective

To create OSS Server, in order to handle the custom created OSS Client.

Tasks

- Create the OSS Server by writing a class OSS_Server.

The Role of Studio

No role for Studio, the above class can be created standalone.

Creating CORBA Interface for OSS

Objective

To create CORBA Interface for OSS, in order to handle the communication between OSS Client and NMS.

Tasks

- Create the CORBA Interface for OSS by writing a class OSSImpl.

The Role of Studio

No role for Studio, the above class can be created standalone.

Common

Creating the IDL common to both NMS and OSS

Objective

To create IDL common to both NMS and OSS.

Tasks

- Create the IDL called OSS.idl.

The Role of Studio

No role for Studio, the above class can be created standalone.
4. Implementation - Web NMS

The Application implementation is in three parts. First part is Web NMS implementation. Next part is common implementation. The last part is OSS implementation.

This section deals with Web NMS implementation in detail.

Create this tutorial application using **AdventNet Web NMS Studio**. This comes bundled along with AdventNet Web NMS.

**Using AdventNet Web NMS Studio**

In **AdventNet Web NMS Studio**, you have to create a separate project for this application. When the project is complete, compile it and package it into a NAR file. For deploying the application in the AdventNet Web NMS, you have to deploy the NAR into the AdventNet Web NMS using the Deployment Wizard tool. Various features available in the Studio allow you to create the Application. However, you need to write certain amount of custom code in order to suit the need of the tutorial application. You need to add certain files as **other files** in the Studio project, which are specific to the exclusive implementation of this tutorial application.

**Overview**

To start with, you have to create a new Studio project.

In the Project, build the application using the following Service Wizards:

**Exposing Web NMS Resources to OSS**

You have to configure the **corba_parameters.conf** file, as explained in the detailed implementation, to expose Web NMS resources to OSS. You can find the configuration file located in the Studio project directory.

**Implementing CORBA Support in Web NMS: Configuring the Process**

You have to configure the custom CORBA process (i.e., **CorbaServer.java**) which you will be creating in the later part of the implementation) in the **NmsProcessesBE.conf** file to register it as one of the AdventNet processes.

You need configure the custom CORBA process to implement CORBA support in Web NMS, as explained in the detailed implementation. You can find the configuration file located in the Server Configuration node of the Studio project.

**Extending the Managed Object**

For modeling a CORBA object, you have to write a Java class extending the Web NMS Managed Object. You have to create a Java file named **ChannelData.java** in the Other files node on the Studio project.

**Implementing CORBA Support in Web NMS: Creating the Process**

You have to write a custom process for CORBA (a Java file). Also you need to configure the process in the NmsProcessesBE.conf file to register it as Web NMS process. You have to create a Java file named **CorbaServer.java** in the Other files node on the Studio project.

**Notifying Web NMS Alerts to OSS**

You have to write Java class to register the CORBA process as Alerts listener. You have to create a Java file named **OSS.java** in the Other files node on the Studio project.
4.1 Creating CNB Project

The first step toward any implementation using the AdventNet Web NMS Studio is to create a Project. The project stores all the information that are essential for developing the application.

For details of Creating Project of AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.

Instructions

Follow the steps given below to create the project.

**Step 1: Invoke the Project Wizard**

Select File > New Project menu item to invoke the Project Wizard.

**Step 2: Add Project Details**

Provide the following details about the CNB Project.

- **Project Name**: CNB User
- **Package Name**: OSSystem
- **Application Name**: CNB User
- **Version**: 1.0

Click Next to proceed.

**Step 3: Select Device Details**

Provide the following details about the Device.

- **Device Type**: CorbaDevice

**Step 4: Select Web NMS Services Details**

Select the Services to be customized in your application. A few Services are selected by default.

For this application, select only the following Services (which includes the default Services):

- MANAGEDRESOURCE
- SECURITY
- SERVER_CONFIGURATION
- REBRANDING

Click Next to proceed.

**Step 5: Select Database Details**

Select the Databases that are listed below:

- MySQL
- Oracle

Click Next to proceed.
**Step 6: Select User Details**

Select *Single User* option.

Click **Next** to proceed.

**Step 7: Preview**

You can view the CNB Project details in the summary screen.

Click **Finish** to create the Project Workspace.

**Result**

The Workspace for the Project is now created. After the Project is created, the Resource Factory starts. Skip the Resource Factory Wizard.
4.2 Configuring CORBA Parameters

To expose Web NMS resources to OSS, you need to enable CORBA protocol in AdventNet Web NMS.

To enable CORBA and change the values of the parameters, you need to specify them in the `corba_parameters.conf` file present in the `<Web NMS HOME>/conf` directory. Studio does not support this file.

- Open the `corba_parameters.conf` file in a basic text editor (Notepad, GVIM etc.). The file is available in the `<Web NMS Home>/StudioTools/Studio/projects/CNB_Tutorial/WebNMS/conf` directory.
- Enter the contents given below into the file.
- Save the file.

**Parameters of corba_parameters.conf file**

In the configuration file, the parameters for the CORBA have to be specified against the keyword `PARAMETERS`.

Make the following entry in the conf file. This will take three parameters EnableCORBA, ORBParms, and RootContextName with values true, `-ORBInitialPort 1050`, and `NameService` respectively.

```xml
<PARAMETERS EnableCORBA="true" OrbParms="-ORBInitialPort 1050"
RootContextName="NameService" />
```

You can also configure `ORBInitialHost` in the `ORBParams`, if the CORBA Service is running in a different machine. Example entry is as given below:

```
-ORBInitialHost <machine name>
```
4.3 Modeling CORBA Object

This section explains you how to model a CORBA object.

To model a CORBA object, you have to write an object class by extending the Web NMS core object (i.e., Managed Object). Write an object class called ChannelData.java.

Create this Java file in the Other files node of the Studio project.

For details of Working with other files of AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.
4.3.1 ChannelData

To implement the ChannelData

1. Define the package

```java
package OSsystem;
```

2. Define the class. It extends the AdventNet Web NMS ManagedObject. It implements the ChannelData_CIOperations, generated by compiling the OSS.idl, for CORBA support.

```java
public class ChannelData extends com.adventnet.nms.topodb.ManagedObject implements OSSystem._ChannelData_CIOperations {
```

3. Define the instance variables

```java
public int currentChannelRate;
...
public int crcBlockLength;
```

4. Define the constructors

```java
public ChannelData() {
}
public ChannelData(int __currentChannelRate, int __previousChannelRate, int __interleaveDelay, int __crcBlockLength) {
  currentChannelRate = __currentChannelRate;
  previousChannelRate = __previousChannelRate;
  interleaveDelay = __interleaveDelay;
  crcBlockLength = __crcBlockLength;
}
```

5. Implement the get method for the ChannelData Object. Similarly, the other three attributes are also implemented.

```java
public int currentChannelRate() {
  return currentChannelRate;
}
```

6. Implement the set method for the ChannelData Object. Similarly, the other three attributes are also implemented.

```java
public void currentChannelRate(int param) {
  currentChannelRate = param;
}
```

Click ChannelData.java to see the complete source file.
4.4 Implementing CORBA Support in Web NMS

Creating the process

This section will guide you through implementing CORBA support in Web NMS.

To implement CORBA support in Web NMS, you have to write a custom process (a Java class) called `CorbaServer.java`. You have to configure this process as explained in the Configuring the process section.

Create this Java file in the Other files node of the Studio project.

For details of Working with other files of AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.

Configuring the Process

To implement CORBA support in Web NMS, you have to write custom process. Make the custom process a plug in module of Web NMS BE server. To plug in your module to BE server, ensure that you implement the `com.adventnet.nms.util.RunProcessInterface` in the process and configure the process in the `NmsProcessesBE.conf` file.

Module Services offers

- Provision to define modules that are individual processes in the Web NMS BE server
- The life-cycle methods for the modules
- Provision to start and stop the modules
- Plugging in your own module into the Web NMS Server.

To plug in your module in the Web NMS servers, you need to configure the module in the `NmsProcessesBE.conf` present in the `<Web NMS HOME>/conf` directory. Specify parameters in the configuration file, which will be passed as arguments to your module when it is started by the Web NMS server.
4.4.1 CorbaServer

To implement the CorbaServer, first you define the package.

```java
package OSSystem;
```

Then, you import the necessary packages.

```java
import com.adventnet.management.log.Log;
... ...
import org.omg.CosNaming.*;
```

Define the class and implement the `RunProcessInterface` so that Web NMS can start the CorbaServer through the `NmsProcessesBE.conf` file.

```java
public class CorbaServer implements RunProcessInterface {
```

The method below is a `RunProcessInterface` method called by Web NMS to optionally run CorbaServer in the same JVM as Web NMS. This runs the CorbaServer, i.e., initializes the ORB and binds the Servant Object Reference in Name Service.

```java
public void callMain(String argv[]) {
    try {
        boolean processState = true;
        String argv[] = {args[2], args[3], args[4], args[5]};
        // create and initialize the ORB
        ORB orb = ORB.init(argv, null);

        // create servant and register it with the ORB
        OSS api = new OSS();
        orb.connect(api);

        // get the root naming context -- Modify the code to get this from the Corba_parameters.conf
        org.omg.CORBA.Object objRef = orb.resolve_initial_references(args[0]);
        NamingContext ncRef = NamingContextHelper.narrow(objRef);

        // bind the Object Reference in Naming
        NameComponent nc = new NameComponent(args[1], "");
        NameComponent path[] = {nc};
        ncRef.rebind(path, api);

    } catch ( Exception re) {
        //re.printStackTrace();
        NmsLogMgr.MISCERR.fail("Exception occured in Northbound CORBA
\n\t::: ", re);
        processState = false;
    }
}
```
The method given below is a `RunProcessInterface` method called by Web NMS to check if the modules have been initialized. If the TNAMESERVER is not started, boolean false will be returned and the server will be shut down automatically.

```java
public boolean isInitialized()
{
  if(! processState)
  {
    try
    {
      System.err.println(" JAVA IDL NAME SERVER IS NOT RUNNING ; NMS SERVER IS SHUTTING DOWN ....");
      ShutDownAPIImpl api = ShutDownAPIImpl.getInstance();
      api.shutDownNMSServer(true);
    }
    catch(Exception e)
    {
      System.out.println(" Exception Caught While Shutting Down the Server ");
    }
    return processState;
  }
}
```

When the Web NMS server is being shut down by the user, the `RunProcessInterface` method given below will be called. So, the implementation of this method could take care to save the state, if any.

```java
public void shutDown()
{
  if (PureServerUtilsBE.rmiBind)
  {
    try
    {
      String url = PureServerUtils.getRMIURL("ORB");
      if (url != null)
      {
        Naming.unbind(url);
        NmsLogMgr.MISCUSER.log("CORBA ORB successfully unbound from registry",Log.SUMMARY);
      }
    }
    catch ( Exception e)
    {
      NmsLogMgr.MISCERR.fail("Exception in unbinding CORBA ORB : " + e, null);
    }
  }
}
```

Click CorbaServer.java to see the complete source file.
4.4.2 Server Configuration

Editing NmsProcessesBE.conf File

The Studio provides a normal text Editor to edit NmsProcessesBE.conf file. The Server Configuration wizard of the Studio provides the facility to create and edit the nodes of the server conf file and its details. You can add or edit the node or modify the existing node directly as per your requirement using the UI. The UI is invoked by the right click menu items Add, Modify, and Delete.

For details of Working with Configuration files of AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.

Parameters of NmsProcessesBE.conf File

In the configuration file, the fully qualified class name of your implementation of com.adventnet.nms.util.RunProcessInterface should be specified against the keyword PROCESS, and the parameters for the module have to be specified against the keyword ARGS.

Make the following entry in the conf file. CorbaServer is the implementation of com.adventnet.nms.util.RunProcessInterface, which belongs to the OSSystem package. This module will take the parameters "NameService" <NameReference>, "-ORBInitialHost" <ORBInitialHost>, and "-ORBInitialPort" <ORBInitialPort>.

The default parameter values are:

<NameReference> - WebNMS
<ORBInitialHost> - localhost
<ORBInitialPort> - 1050

```java
#java OSSystem.CorbaServer NameService WebNMS -ORBInitialHost
nameserverhost -ORBInitialPort nameserverport
PROCESS OSSystem.CorbaServer
ARGS NameService WebNMS -ORBInitialHost localhost -ORBInitialPort 1050
```
4.5 Notifying Web NMS Alerts to OSS

This section will tell you how to notify Web NMS Alerts to OSS.

To notify Web NMS Alerts to OSS, you have to write a Java class called **OSS.java**. Ensure that the **OSS.class** extends **NorthBoundInterfaceImplBase** and implements AlertObserver. As a result, the OSS class registers itself as an Alert Observer. It gets the reference to client side class **OSSImpl** when the client (OSS_Server) is started. Using this callback object reference, **OSS.class** sends alerts to OSS_Server through CORBA callbacks.

Create this Java file in the **Other files** node of the Studio project.

For details of **Working with other files** of AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.
4.5.1 OSS

To implement the OSS Web NMS side CORBA servant, first you define the package.

```java
package OSSSystem;
```

Then, you import the necessary packages.

```java
import com.adventnet.nms.util.NmsLogMgr;
... ... ...
import OSSystem.*;
```

Then, you define the OSS class which is the API interface for accessing, updating the Northbound CORBA Objects in the device. It extends _NorthBoundInterfaceImplBase (the CORBA class generated by compiling the oss.idl) for Northbound CORBA support. It implements AlertObserver to register itself with Web NMS. Whenever there is an addition, update, or deletion of Alert Object takes place within the Web NMS System, Web NMS intimates OSS by calling `update(Alert alert)` method.

```java
public class OSS extends OSSystem._NorthBoundInterfaceImplBase implements AlertObserver
{
    private TopoAPI api = null;
    private NorthBoundInterface callback = null;

    OSS()
    {
        super();
        while( api == null)
        {
            try
            {
                api = com.adventnet.nms.util.NmsUtil.getTopoAPI()Thread.sleep(1000);
            }
            catch ( Exception e)
            {
                e.printStackTrace();
            }
        }
    }

    The following CORBA method gets the AdventNet ManagedObjects from the Web NMS Server.

```java
public ManagedObject_CI getManagedObject(ManagedObject_CI MO_TREE)
{
    try
    {
```
Get the MO based on the properties/attribute values on MO_TREE

```java
ManagedObject MO = api.getByName(MO_TREE.getName());
ManagedObject_CI mo = new _ManagedObject_CITie(MO);
return mo;
}
```

```java
The following CORBA method gets the ChannelData from the Web NMS Server for a particular channel

```java
public ChannelData_CI getChannelData(String channelID) {
    try {
        ChannelData_CD = new ChannelData(1,2,3,4);
        api.getObject("channeldata", channelID);
        ChannelData_CI cd = new _ChannelData_CITie(CD);
        return cd;
    }
    catch (Exception e) {
        NmsLogMgr.MISCERR.fail("From NorthBoundCorbaServer:Unable to getChannelData from the server.", e);
        return null;
    }
}
```

The following CORBA method is used for Provisioning/Configuring the device. To configure the device, set the AdventNet ManagedObjects properties on Web NMS server.

```java
public boolean setManagedObject(ManagedObject_CI MO_TREE) {
    try {
        ManagedObject MO = api.getByName(MO_TREE.getName());
        .
        .
        MO.setStatus(state);
        .
        .
        return api.updateObject(MO, false, false);
    }
    catch (Exception e) {
        NmsLogMgr.MISCERR.fail("From NorthBoundCorbaServer:Unable to setManagedObject to the WebNMS server.", e);
        return false;
    }
}
```
The following CORBA method is for Fault Management to register OSS's `alarmAPI` for callback.

```java
public boolean registerAlarmCallback(NorthBoundInterface cb) {
    try {
        AlertAPI api = com.adventnet.nms.util.NmsUtil.getAlertAPI();
        boolean res = api.registerForAlerts(this);
        callback = cb;
        return true;
    } catch( Exception err ) {
        NmsLogMgr.MISCERR.fail("Registration for alert observer failure.", err);
        return false;
    }
}
```

The following method implements AlertObserver. Whenever there is an addition, update, or deletion of Alert Object in Web NMS, these methods are called. These methods in turn inform OSS Application through CORBA callback mechanism.

```java
public void update(Alert a) {
    try {
        Alert_CI aci = new _Alert_CITie(a);
        if (callback != null) this.storeAlarm(aci);
    } catch( Exception err ) {
        NmsLogMgr.MISCERR.fail("Callback for Alarm failure.", err);
    }
}

public void storeAlarm(Alert_CI a) {
    callback.storeAlarm(a);
}
```

Click OSS.java to see the complete source file.
4.6 Packaging the Project

This topic explains adding the custom classes, compiling, and packaging the Studio project.

- Adding the Custom Classes
- Compiling the Project
- Packaging the Application

Adding the Custom Classes

Add the custom classes to enable CORBA data transaction on the Web NMS side. These classes are generated from the IDL to Java conversion of a IDL file (OSS.idl) written for this purpose. Explanation about writing the IDL file and converting it into Java files are dealt in the Implementation - Common section of this document.

- Copy the compiled class files from the IDL to Java conversion location into the classes directory of the Studio project directory.

<Web NMS HOME>/StudioTools/Studio/projects/CNB_Tutorial/classes

Compiling the Project

After adding the classes, compile the project.

- Select the Project > Compile menu from the Studio.

Packaging the Application

After compiling the project, package the application into a NAR file.

- Select the Project > Package menu from the Studio.

Package Wizard comes up and carry out the packaging in 8 steps.

1. **Project and NAR Details** - Enter the Nar File Name as CNB_User.nar, Destination Directory, and README File Name and click Next button.
2. **Select Services** - Select the Services you have configured so far in the project and click Next button.
3. **Database Details** - Select the database. Enter the database details URL, Driver Name, User Name, Password, Trans Connection, and Non-Trans Connection and click Next button.
4. **Map and List Icons** - Select the Map Icon and List Icons for various severity and click Next button.
5. **Other Resources** - Select the menus and images need to bundled in the NAR. In the Resource Path, select the corba_parameters.conf and in the Target Directory enter ./conf. Click Add button. Click Next button.
6. **Additional JARs** - Add the additional JARs required and click Next button.
7. **Client NARs** - Add the required Client NARs imported using Import NAR Wizard and click Next button.
8. **Create NAR** - Select the packaging option as required and click Finish button.

For details of Packaging Studio Applications of AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.

This will generate a NAR file called CNB_User.nar. You can deploy this NAR directly into Web NMS to install the application.
5. Common Implementation

This section deals with the second part of the implementation (i.e., common implementation).

To envisage data transaction in CORBA protocol between NMS and OSS, you have to create a custom IDL named *OSS.idl*. In this IDL, you will be using the Web NMS IDLs. The details are explained in the following topic.

After creating the *OSS.idl*, you will be converting IDL to Java file. This conversion will generate a set of Java source files. These Java files are compiled and the class files are available in the package structure *OSSystem*. They are copied into two locations. One is in the *classes* directory of the Studio project for Web NMS implementation and the other is in the *classes* directory of the Bean Builder project for OSS implementation.
5.1 Creating Custom IDL for OSS-NMS Data Transaction

This topic will tell you how the Web NMS IDLs are used to build a Northbound CORBA application. It will explain how you can write your own IDLs for application extensions. You can then develop the application examples using the IDL interfaces.

Creating OSS.idl file

1. Open a basic text editor (Notepad, VIM etc.).
2. Copy the OSS.idl file from <Web NMS Home>/tutorials/corba_northbound_tutorial/idls directory.
3. Save the file.

Once you have the OSS.idl, the next step is to generate the Java code for the OSS.idl using idltojava compiler. You get all the necessary Java files by executing the following command. Make sure that your path is set to your <Java Home>/bin directory.

```
>idltojava -fno-cpp -ftie OSS.idl
```

⚠️ **Warning:** The above command can be executed only in JDK Version 1.2. All other Java compilations etc. can be done only in JDK Version 1.3

The following are the custom source files generated by compiling the OSS.idl using idltojava

- NorthBoundInterface
  - NorthBoundInterfaceImplBase.java
  - NorthBoundInterfaceOperations.java
  - NorthBoundInterfaceStub.java
  - NorthBoundInterfaceTie.java
  - NorthBoundInterface.java
  - NorthBoundInterfaceHelper.java
  - NorthBoundInterfaceHolder.java

- ChannelData_CI
  - ChannelData_CIImplBase.java
  - ChannelData_CIOperations.java
  - ChannelData_CIDStub.java
  - ChannelData_CIDTie.java
  - ChannelData_CI.java
  - ChannelData_CIHelper.java
  - ChannelData_CIHolder.java

The OSS.idl is built using the AdventNet Web NMS IDLs - Common.idl, Topology.idl, and Alert.idl.

The OSS.idl defines attributes, interfaces and methods required for providing Northbound support. You define the ChannelData_CI interface for a Channel Data object. The Channel Data object, which extends the Managed Object, will store some additional attributes specific to the channel. You then define NorthBoundInterface for methods to get/set Managed Objects and to send Alarms from Web NMS to OSS using callback. The topic OSS.idl in the sub section discusses how to write the OSS.idl in more detail.
5.1.1 OSS.idl

You have to extend the Web NMS IDL files in the new IDLs to use the module API's of Web NMS. This is done in **OSS.idl** which uses **Common.idl**, **Topology.idl**, and **Alert.idl** and adds a custom ManagedObject and few more custom methods. To do this, first you define the module OSSystem.

```plaintext
module OSSystem {

Then, you define the ChannelData_CI extending the ManagedObject_CI. This creates a ChannelData_CI Object which extends the ManagedObject. The ChannelData_CI Object is passed from Web NMS to OSS Application when the `getChannelData` method is invoked by the OSS.

```plaintext
interface ChannelData_CI :
com::adventnet::nms::topodb::corba::ManagedObject_CI {

attribute long currentChannelRate;
attribute long previousChannelRate;
attribute long interleaveDelay;
attribute long crcBlockLength;
}
```

Then, you define the attributes of the ChannelData Object.

```plaintext
interface NorthBoundInterface{

The following method gets MO from Web NMS based on MO passed from the OSS module.

```plaintext
com::adventnet::nms::topodb::corba::ManagedObject_CI getManagedObject(in com::adventnet::nms::topodb::corba::ManagedObject_CI MO);

The following method gets ChannelData from Web NMS based on Channel ID passed from the OSS module.

```plaintext
OSSystem::ChannelData_CI getChannelData(in string channel);

The following method sets MO Web NMS based on MO passed from the OSS module.

```plaintext
boolean setManagedObject(in com::adventnet::nms::topodb::corba::ManagedObject_CI MO);

The following method registers the AlertCallback with the Web NMS server for alertCallback. Whenever an alert is created/updated/deleted, the OSS system will get called.

```plaintext
boolean registerAlarmCallback(in NorthBoundInterface cb);

You include **Alert.idl** (Web NMS) to pass `com.adventnet.nms.alaertdb.alert` to and fro OSS system and Web NMS.

```plaintext
void storeAlarm(in com::adventnet::nms::alaertdb::corba::Alert_CI alarm);
}
```

The above methods are implemented on the server, i.e., the Web NMS side by **oss.java** and **ChannelData.java**.

On the client side, i.e., the OSS side, the CORBA callback methods are implemented by **OSSImpl.java**.

You can click the hyperlink to view the complete OSS.idl.java.
6. OSS Implementation

This section deals with the last part of the implementation (i.e., OSS implementation).

- Invoking the Bean Builder tool
- User Interface Design
- Writing OSS Client classes

In this implementation, first you will be creating the User Interface (UI) screens for the OSS Client. For this you will be using Bean Builder tool of AdventNet Web NMS.

Invoking the Bean Builder tool

You can invoke the Bean Builder tool in two ways.

Run the BeanBuilder.bat/sh file from <Web NMS Home>/bin directory.

[OR]

Alternatively, double-click the Bean Builder icon in the Web NMS Launcher > Web NMS IDE.

User Interface Design

The design of User Interface of the OSS Client is as follows:

The left pane of the UI display the OSS Client Panel Tree. The tree comprises of

- One Main Screen called CNB.
- Three Panel Screens

The three panel screens are

1. MO
2. Alarms
3. Events

The MO Screen will list all the Managed Objects of Web NMS in a tabular format.

The Alarms Screen will list all the Alerts of Web NMS in a tabular format.

The Events Screen will list all the Events of Web NMS in a tabular format.

Double-click on any Managed Objects in the MO Panel Screen. Status screen will pop up.

The Status Screen contains three tab screens.

1. MO Status
2. Alert Messages
3. Channel Messages

The MO Status Tab displays the status of the selected Managed Object and allows to configure the same.

The Alert Message Tab displays the Notifications from MO. This screen will get updated with any change in status of Web NMS Managed Objects.

The Channel Message Tab displays the Channel Details which is set in Web NMS.
Writing OSS Client classes

Write two classes in the Bean Builder project itself as Java files. The two files are OSS_Server and OSSImpl. The OSS_Server class is the standalone OSS and the OSSImpl class is a client side Servant implementation.
Finally, integrate the IDL generated Java classes into the Bean Builder project.

Compile the Bean Builder project and use the compiled classes as OSS.

Apart from this, write a batch/shell file to invoke the OSS.
6.1 Opening the Bean Builder Project

You have to build the OSS Client using the Bean Builder of Web NMS. Follow the procedure given below to build the OSS Client:

Instructions

Follow the steps given below to open the project.

**Step 1: Open the Bean Builder Project**

Select File > Open > Project menu item to open the Project.

Open Project dialog box will pop up.

**Step 2: Select the Project**

Select the Project **OSS_Client.proj** from the `<Web NMS Home>/tutorials/corba_northbound_tutorial/OSS/projects` directory.

Click Open to proceed.

**Step 3: View**

You can view the CNB OSS Client Project details in the screen.

Result

The Project is now open. After opening the Project, create the OSS Client classes and get the IDL to Java files copied into the Project space. The procedure is given in the following topics.
6.2 Creating Operational Support System: OSS_Server

The OSS_Server is a standalone program representing the OSS.

1. Create this class in the project.
2. Invoke the File > New > File Menu item.
3. The New File dialog box will pop up.
   In the General tab, enter the following details.
   - File Name as OSS_Server.java.
   - Package as OSSystem.
   - Leave the Extends field blank.
   - Leave the Directory unchanged.
4. Click OK. This will open the Java file in the Project.
5. Copy the contents of the OSS_Server.java file to this file.

The description of the code highlight is given below.

OSS_Server

To implement the OSS_Server, first you define the package and import the necessary packages. After that, you define the OSS_Server class.

```java
public class OSS_Server
{
    public static void main(String[] argv)
    {
        try
```

The arguments will be like NameService Web NMS, -ORBInitialHost localhost, -ORBInitialPort 1050.

```java
    String args[] = {argv[2], argv[3], argv[4], argv[5]};
```

Then, you initialize the ORB and resolve the Object Reference in Naming.

```java
    ORB = ORB.init(args, null);
    org.omg.CORBA.Object objRef = orb.resolve_initial_references(argv[0]);
    NamingContext ncRef = NamingContextHelper.narrow(objRef);
    NameComponent nc = new NameComponent(argv[1], "");
    NameComponent path[] = {nc};
    NorthBoundInterface webNMSRef = NorthBoundInterfaceHelper.narrow(ncRef.resolve(path));
```

Then, you register for Alarm callbacks from the Web NMS.

```java
    oss = new OSSImpl();
    orb.connect(oss);
    oss.registerAlarmCallback(webNMSRef);
```

To get the handle of Topo API, Event API, and Alert API to operate on the NMS data, you have to add the following piece of code.
NameComponent nc1 = new NameComponent("com","com");
NameComponent nc2 = new NameComponent("adventnet","adventnet");
NameComponent nc3 = new NameComponent("nms","nms");
NameComponent[] nmsNC = { nc1, nc2, nc3};
NamingContext nmsContext =
NamingContextHelper.narrow(ncRef.resolve(nmsNC));
NameComponent nc4 = new NameComponent("TopoAPI", "");
NameComponent[] name = {nc4};
// resolve the Object Reference using NamingService
org.omg.CORBA.Object objectRef = nmsContext.resolve(name);

// Obtain the TopoAPI_CI ref from it
topo_api = TopoAPI_CIHelper.narrow(objectRef);
nc4 = new NameComponent("EventAPI",""lw);
name[0] = nc4;
event_api = EventAPI_CIHelper.narrow(nmsContext.resolve(name));
nc4 = new NameComponent("AlertAPI",""lw);
name[0] = nc4;
alert_api = AlertAPI_CIHelper.narrow(nmsContext.resolve(name));

Then, you will have to develop an OSS Client UI for manipulating the NMS data. In this UI, all the MOs, Events, and Alerts will be shown.

new MainScreen();

Click OSS_Server.java to see the complete source file.
6.3 CORBA Interface for OSS: OSSImpl

OSSImpl class is the client side servant implementation.

1. Create this class in the project.
2. Invoke the File > New > File Menu item.
3. The New File dialog box will pop up.
   In the General tab, enter the following details.
   - **File Name** as OSSImpl.java.
   - **Package** as OSSystem.
   - **Extends** as OSSystem._NorthBoundInterfaceImplBase.
   Leave the Directory unchanged.
   In the Interfaces tab, enter the following details.
   Leave the Interfaces field blank.
4. Click OK. This will open the Java file in the Project.
5. Copy the contents of the OSSImpl.java file to this file.

The description of the code high light is given below.

**OSSImpl**

To implement the CorbaServer, first you define the package and import the necessary packages. After that, you define the OSSImpl class. This is the API interface for accessing, updating the North bound CORBA Objects in the device through Web NMS

```java
public class OSSImpl extends OSSystem._NorthBoundInterfaceImplBase
{
    private NorthBoundInterface webNMSRef = null;
}
```

To get the AdventNet ManagedObjects from the Web NMS Server, you define the following method.

```java
public ManagedObject_CI getManagedObject(ManagedObject_CI MO_TREE)
{
    try
    {
        return webNMSRef.getManagedObject(MO_TREE);
    }
    catch (Exception e)
    {
        System.out.println("From NorthBoundCorbaServer:Unable to getManagedObject from the server." + e);
        return null;
    }
}
```

The following portion of the code gets the MO based on the properties/attribute values on MO_TREE.

Then, you get the ChannelData from the Web NMS Server (for a particular channel) and MO based on the properties/attribute values on MO_TREE.
public ChannelData_CI getChannelData(String channelID) {
    try {
        return webNMSRef.getChannelData(channelID);
    } catch (Exception e) {
        System.out.println("From NorthBoundCorbaServer:Unable to getChannelData from the server." + e);
    }
    return null;
}

In this method, you set the AdventNet ManagedObjects properties on Web NMS server for Provisioning/Configuring the device.

public boolean setManagedObject(ManagedObject_CI MO_TREE) {
    try {
        return webNMSRef.setManagedObject(MO_TREE);
    } catch (Exception e) {
        System.out.println("From NorthBoundCorbaServer:Unable to setManagedObject to the WebNMS server." + e);
    }
    return false;
}

This method registers OSS's alarmAPI for callback for Fault Management.

public boolean registerAlarmCallback(NorthBoundInterface cb) {
    try {
        webNMSRef = cb;
        return webNMSRef.registerAlarmCallback(this);
    } catch( Exception err ) {
        System.out.println("Registration for alert callback failure." + err);
    }
    return false;
}

public void storeAlarm(Alert_CI a) {
    jta.append("Alert message from WebNMS: " + a.getSource());
    jta.append("Alert message from WebNMS: " + a.getMessage());
    jta.append("Alert Severity from WebNMS: " + a.getSeverity());
}

Click OSSImpl.java to see the complete source file.
6.4 Compiling the Project

This topic explains adding the custom classes and compiling the Bean Builder project.

- Adding the Custom IDL Classes
- Compiling the Project

Adding the Custom IDL Classes

Add the custom classes to enable CORBA data transaction on the OSS side. These classes are generated from the IDL to Java conversion of IDL file (OSS.idl) written for this purpose. Explanation about writing the IDL file and converting it into Java files are dealt in the Implementation - Common section of this document.

- Copy the compiled class files with OSSystem package structure, from the IDL to Java conversion location into the Bean Builder project directory location given below.

    <Web NMS HOME>/tutorials/corba_northbound_tutorial/oss/projects/OSS_Client/classes

Compile the Project

After adding the classes, compile the project.

- Select the Project > Compile Project menu from the Bean Builder.

This will compile the classes of the project and application is ready.
7. Fast Track Implementation

After the building of services is complete, compile the project and package it as a NAR file.

This topic will guide you through shortcut to the tutorial, if you are unable to complete the Tutorial Project.
It will also guide you, if you want to try quick modifications in the available project and check the results.

Successful completion of Project

After the building of application is complete, compile the project and package it as a NAR file. If you are able to create the NAR successfully then skip this topic.

Try your requirements quickly in the Project

If you want to try your requirements in the Studio project on your own efforts, then you can use the Studio project of this application which is bundled in AdventNet Web NMS Studio.

The Studio project of the application comes bundled in AdventNet Web NMS Studio in the following URL:

<Web NMS Home>/StudioTools/Studio/projects/CNB_Tutorial.proj

In this you have two options:

1. Compiling without modifications.
2. Compiling with modifications.

First option

- Open the Project in AdventNet Web NMS Studio.
- Compile the Project.
- Package the Project as (CNB_Tutorial1.1.nar) NAR file.

Second Option

- Open the project in AdventNet Web NMS Studio.
- Carryout the Studio supported changes in the project.
- Compile the Project.
- Package the Project as (CNB_Tutorial1.1.nar) NAR file.

- Using the instructions given in the Installing the application section, deploy the CNB_Tutorial1.1.nar file.
- Run the Application.
- View the Result. If you have modified the Project, test the application for the desired result.

Note: If you find difficulty in opening the Project, in the Project Properties check Web NMS Home in the General tab screen and Class Path of various JAR files in the Compiler tab screen. These will be having the values in which they were developed. Change both the parameters with respect to your Web NMS Installation.
8. Deployment and Testing

8.1 Installation Notes

This application is bundled with Web NMS 4.

The CORBA Northbound Tutorial is present in <Web NMS HOME>/tutorials/corba_northbound_tutorial directory.

You can also download this tutorial from the AdventNet Web site. Please use the following URL to download the tutorial.

http://download.adventnet.com/products/webnms/tutorials/corba_northbound_tutorial.zip

If you are using the downloaded tutorial from the Web site, then extract the tutorial under <Web NMS HOME> directory.

Installing the Application

To install the tutorial application follow the steps given below.

- Stop the Web NMS Server if it is running.
- Run the DeploymentWizard.bat/.sh file under <Web NMS HOME>/bin directory and also ensure that the server is not running.
- Select the NarInstall/Uninstall tab.
- Select Install button in the screen.
- Click Browse button in the NAR INSTALLER pop-up screen.
- Select the tutorials/corba_northbound_tutorial/nms/CNB_User.nar from the File Chooser pop-up screen and click the Select button.
- Click OK button in the NAR INSTALLER pop-up screen.
- On clicking OK button, NmsPwsNarInstaller screen will pop up.
- In that, Select the Application Database (in which you want to run the tutorial) listed from the combo box.
- Click Next button.
- In this screen, click Install button.
- This will install and show the status in the progress bar. See that the progress is 100%.
- Click Close button. You will see CNB_User.nar entry listed in the NarInstall/Uninstall tab.
- Now the installation is complete.
- Click Exit button to quit the Deployment Wizard.

- Reinitialize Web NMS using initialize_nms.bat/.sh. Select Both Data and Configuration option in the appearing form for reinitializing Web NMS.

Caution:
This tutorial will function only if Web NMS is reinitialized. Hence, after installing the tutorial, reinitialize Web NMS. Ensure that you use separate Web NMS installation for tutorial application development and installation. Do not experiment on Web NMS deployed in real world.

When this tutorial application is installed, the existing files which require changes from the installed Web NMS are backed up. However, when you uninstall this tutorial application, the files modified will be restored from the backup.
Uninstalling the Application

To uninstall the tutorial application follow the steps given below.

- Run the `DeploymentWizard.bat/.sh` file under `<Web NMS HOME>/bin` directory.
- Select the NarInstall/Uninstall tab.
- In the Uninstall section
- Select the `CNB_Tutorial1.0.nar` entry
- Click Uninstall button in the screen.
- On clicking Uninstall button, `NmsPwsNarUninstaller` screen will pop up.
- In that, click Uninstall button.
- This will uninstall and show the status in the progress bar. See that the progress is 100%.
- Click Close button.
- On clicking Close button, you will see Uninstalled Successfully pop-up message with an OK button.
- Click OK button.
- Now the uninstallation is complete.
- Click Exit button to quit the Deployment Wizard.

Modifying the Application

If you want to modify the application and try out other features, you can edit the Studio project of this application bundled with this tutorial.

You can open the project's `CNB_User.proj` file from the `<Web NMS HOME>/Studio_Tools/studio/projects` directory, in the AdventNet Web NMS Studio.

Make the changes (supported by the AdventNet Web NMS Studio) to the project as per your requirement.

Save the changes and Compile the project and package it into NAR.

Now, uninstall the existing installed NAR and install the new NAR generated with your changes.

Caution: Uninstalling the original NAR is mandatory as only one Studio NAR is permitted to be deployed in AdventNet Web NMS.

For complete details of Working with AdventNet Web NMS Studio, refer to the AdventNet Web NMS Studio documentation.
8.2 Testing

Start the Application

Starting the JAVA IDL Name Service

- From the command prompt, start the JAVA IDL name server by executing the following command making sure that the path is set to JDK1.3.

>tnameserv -ORBInitialPort <port number>

**Note:** Default port number recommended for JAVA IDL Name Service for Web NMS CORBA applications is 1050.

Starting the Web NMS

- Start the Web NMS Server by executing the `startnms.bat/.sh` script in `<WebNMSHome>/bin` directory.

**Note:** The host name and port number in `NmsProcessesBE.conf` and `corba_parameters.conf` should be same as that of Java IDL Name Server. By default, the `hostname` is set as `localhost` and `port number` as 1050.

Starting the OSS Client

- Ensure that the classpath is set to `</WebNMSHome>/StudioTools/Studio/projects/CNB_Tutorial/classes, <WebNMSHome>/classes, <WebNMSHome>/classes\crimson.jar and <WebNMSHome>/classes\ApiUtils.jar`

**Note:** The above classes and jar file are needed to create the AdventNet Web NMS ManagedObject in OSS Client.

- Run the OSS Client by executing `run_oss_client.bat/sh` file present under `<Web NMS HOME>/ tutorials/corba_northbound_tutorial/oss/bin` directory.

**Note:** The host name and port number in arguments in `run_oss_client.bat/sh` file should be same as that of java IDL Name Server.

- The following UI will be shown. In this UI, the select MO for which you wish to change the status. Double-click on the MO.
The following Status screen will be displayed. Change the status selecting from the combo box, click Set. You can see the status change (before and after setting) visually in the text boxes.
You can see the alert notifications in the following UI. The alert notifications are printed in the text area in the "Alert Messages" tabbed pane, whenever an alarm is created, deleted, or modified in the Web NMS side.

Fig.1 MO Status Update
Fig. 2 Alert Notifications through Callback.

- You can see the Channel-related messages in the following UI.
Testing

Setting the Managed Object Properties with OSS Client

- Start the WEB NMS Application Client by running the `startApplicationClient` batch/shell file present in `<WebNMSHome>/bin` directory. Click your local network map to display your network. Please note the status of the Managed Object representing your Network Element (color of the LED in case of node or color of link in case of interface object that you set in `OSS_Server.java`). You can also right-click the Map Icon corresponding to your network element and check for ManagedObject properties which displays the Status of the Managed Object.
- Note that the LED in case of the node or link in case of interface ManagedObject which was set in the text field changes color/status depending on the status set in ComboBox. (If status is set to 1, it changes to red and status is reported as critical.)
- Now, set the status back to 5 in the ComboBox. The color changes to green and status is reported as clear.

Fig. 3 Channel-related Messages.
Getting the Channel Attributes

- Channel attributes can be set in the OSS.java
  (By default the currentChannelRate = 1, previousChannelRate = 2, interleaveDelay = 3, crcBlockLength = 4).
  Set some other values for the above, say 5,6,7,8. as follows:
  ChannelData CD = new ChannelData(5,6,7,8);
- Build the tutorial,
- Stop the Web NMS if it is running,
- Uninstall the tutorial if it is already installed
- Install the newly built tutorial
- Run the Web NMS
- Run the OSS client.
- You will see the new channel data as shown below.

![Channel-related Messages after Making the Change in OSS.java](image)

Getting the Web NMS Alerts (Alarms) through callbacks

- Start the WEB NMS Application Client and open the Alarms view under Fault Management.
- Run the OSS client.
- Note that the Alerts which are reported in Web NMS Application Client, will also be reported in OSS client which is shown in Fig.2.
8.3 Channel Data Simulation

By default, the Channel attributes have been set in the OSS.java file as follows.

- currentChannelRate = 1
- previousChannelRate = 2
- interleaveDelay = 3
- crcBlockLength = 4

You can modify these values used to construct the ChannelData in the following line in OSS.java

```java
ChannelData CD = new ChannelData(1,2,3,4);
```

If you want to change the Channel data and see the result.

- Stop the Web NMS Server, if it is running.
- Uninstall application, if anything is installed.
- Open the project.
- Open the OSS.java file.
- Make the changes given in the above line.
- Package it as NAR.
- Install this new NAR.
- Reinitialize Web NMS.
- Run Web NMS and OSS Client again.
9. Known Issues

- AdventNet Web NMS uses idltojava compiler to compile all its IDL files and generates the corresponding classes for CORBA support. This tutorial application uses idltojava to compile the OSS.idl so that the generated classes are completely compatible with Web NMS classes.
- Web NMS runs on JRE1.3. To be completely compatible, you should run the Java IDL name server in JDK1.3 (check your path that JDK1.3 is set first and not JDK1.2) Similarly, the OSS should also be run using JDK1.3. If you are using JDK1.3, make sure the JAVA_HOME is set to JDK1.3.
### 10. Trouble Shooting Tips

This topic lists out some of the difficulties that you may face when you develop your own applications or when you run this tutorial.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Reason</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the Web NMS Server is started, the following error message appears: CORBA - unable to resolve name service</td>
<td>The naming service of Java JDK 1.3 may not be running.</td>
<td>Start the naming service of Java. Execute <code>tnameserv.exe</code> in the <code>&lt;Java Home&gt;/bin</code> directory.</td>
</tr>
<tr>
<td>When the Web NMS Server is started, the following exception is thrown: Exception in thread &quot;main&quot; java.lang.NoClass DefFoundError: com/adventnet/nms/topodb/ManagedObject</td>
<td>The Classpath may not be set to <code>&lt;Web NMS Home&gt;/classes</code>.</td>
<td>Set the Classpath to <code>&lt;Web NMS Home&gt;/classes</code>.</td>
</tr>
<tr>
<td>When the Web NMS Server is started, the following exception is thrown: Exception in thread &quot;main&quot; java.lang.NoClass DefFoundError:org/xml/sax/SAXException</td>
<td>The Classpath may not be set to <code>&lt;Web NMS Home&gt;/classes/crimson.jar</code>.</td>
<td>Set the Classpath to <code>&lt;Web NMS Home&gt;/classes/crimson.jar</code>.</td>
</tr>
<tr>
<td>When the OSSClient is started, the following error message appears: java.lang.ArrayIndexOutOfBoundsException</td>
<td>Arguments given while running the OSS Client may be wrong.</td>
<td>Give all the arguments correctly when you run the OSS Client.</td>
</tr>
<tr>
<td>When the OSSClient is started, the following error message appears: org.omg.CosNaming.NamingContextPackage. NotFound</td>
<td>There may be mismatch of naming service source specified, between Web NMS and OSS.</td>
<td>Make sure that the run_oss_client.bat/sh file's second argument (Web NMS) for OSSSystem.OSS_Server is same (check also for the case) as the second argument for OSSSystem.CorbaServer in the NmsProcessesBE.conf file.</td>
</tr>
<tr>
<td>When the OSSClient is started, the following error message appears: org.omg.CORBA.MARSHAL: minor code: 0 completed:</td>
<td>There may be incompatibilities in classes. This can happen if you have made changes in OSS.</td>
<td>Make sure that you have installed the newly built tutorial on Web NMS and using the newly built OSS Client.</td>
</tr>
</tbody>
</table>
## Problem

Client and built the tutorial. If it is not reinstalled, Web NMS will have an older version of classes installed and you would be running a later version of OSS Client. In such a scenario, when OSS client tries to invoke methods on OSS server, then the type checks which occur may throw such exceptions.

### When the Web NMS Server is started, the following error message appears and the server quits.

*Exception during initialisation of CORBA Parameters Server quits*

- **Reason**: JAVA IDL Name Service is not running.
- **Solution**: Start the Java IDL Name Server.
  - Execute the following command:
    ```bash
    <JDK Home>/bin/tnameserv -ORBInitialPort 1050
    ```

### When the OSSClient is started, the following exception is thrown:

*Exception in thread "main" java.lang.NoClassDefFoundError: OSSystem/OSS_Server*

- **Reason**: Classpath not set properly.
- **Solution**: Edit the `run_oss_client.bat` file and include `
  ./projects/OSS_Client/classes` in the Classpath.

### When the OSSClient is started, the following exception is thrown:

*Exception in thread "main" java.lang.NoClassDefFoundError: com/adventnet/nms/eventdb/corba/Event API_CIHelper at OSSystem.OSS_Server.main(OSS_Server.java:90)*

- **Reason**: Classpath not set properly.
- **Solution**: Edit the `run_oss_client.bat` file and include `../../../../../classes` in the Classpath.

### When the OSSClient is started, the following exception is thrown:

*Exception in thread "main" java.lang.NoClassDefFoundError: com/adventnet/apiutils/ParameterChangeListener*

- **Reason**: Classpath not set properly.
- **Solution**: Edit the `run_oss_client.bat` file and include `../../../../../classes/ApiUtils.jar` in the Classpath.

### The OSSClient is running in same machine, where Java IDL Name Server is also running.

When the OSSClient is started, the following exception is thrown:

*org.omg.CORBA.COMM_FAILURE: vmcid: SUN minor code: 201 completed: No*

- **Reason**: Host Name or Port No mismatch.
- **Solution**: The Host Name and Port Number in the arguments of `run_oss_client.bat/sh` file should match with that of Java IDL Name Server.

### The OSSClient is not running in same machine, where Java IDL Name Server is running.

When the OSSClient is started the

- **Reason**: Host Name or Port No mismatch.
- **Solution**: Edit the `run_oss_client.bat/sh` and the change the `localhost` to `<the machine in Java IDL Name Server is running>`.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Reason</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>following exception is thrown: <code>org.omg.CORBA.COMM_FAILURE</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>When the OSSClient is started, the following exception is thrown: <code>org.omg.CosNaming.NamingContextPackage.NotFound: IDL:omg.org/CosNaming/NamingContext/NotFound: 1.</code></td>
<td>CORBA might have not been enabled in <code>corba_parameters.conf</code> file.</td>
<td>To enable CORBA, edit the <code>corba_parameters.conf</code> file in the <code>&lt;Web NMS Home&gt;/conf</code> directory. Set the value for Enable CORBA parameter to <code>true</code>.</td>
</tr>
<tr>
<td>When the OSSClient is started, the following exception is thrown: <code>java.lang.NullPointerException</code> at <code>OSSystem.OSSImpl.registerAlarmCallback(OSSImpl.java:74)</code> at <code>OSSystem.OSS_Server.main(OSS_Server.java:71)</code></td>
<td>Web NMS Server is not running.</td>
<td>Ensure that the OSSClient is started, after starting the Web NMS Server.</td>
</tr>
<tr>
<td>When the Java IDL Name Server is started, the following exception is thrown: <code>TransientNameServer: caught an exception while starting the bootstrap service on port 1050</code></td>
<td>Java IDL Name Server is already running.</td>
<td>If the Java IDL Name Server is already running and the Java IDL Name Server is started, the exception will be thrown.</td>
</tr>
<tr>
<td>License for CORBA module Northbound is not there.</td>
<td>Get the appropriate License with CORBA module Northbound enabled.</td>
<td></td>
</tr>
</tbody>
</table>

AdventNet Inc.
11. Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>OSS</td>
<td>Operational Support Systems</td>
</tr>
<tr>
<td>MO</td>
<td>ManagedObject</td>
</tr>
</tbody>
</table>
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 of 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 working 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.

Building an EMS with CORBA as Southbound Interface

This tutorial explains how developer can build Element Management Systems, which support CORBA as the southbound interface. In this tutorial, we have used TR-005 and TR-035 standards specified by the ANSI T1M1 forum. It explains, how developer can implement the Working Text, WT-046 Version 3.0 as specified in the DSL forum.

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.