Oracle Utilities Network Management System

Configuration Guide

Release 1.11.0.3

E35560-01

August 2012

Oracle Utilities Network Management System Configuration Guide, Release 1.11.0.3

E35560-01

Contents

Preface

Please read through this document thoroughly before beginning your product implementation The purpose of this guide is to provide implementation guidelines for a standard Oracle Utilities Network Management System implementation. This document discusses installation, interfaces, modeling, and software configuration that are considered typical and acceptable for a standard product implementation.

Audience

This document is intended for anyone responsible for the implementation of Oracle Utilities Network Management System.

Related Documents

- Oracle Utilities Network Management System Installation Guide
- Oracle Utilities Network Management System Adapters Guide
- Oracle Utilities Network Management System User's Guide

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Chapter 1

System Overview

The Oracle Utilities Network Management System is an operations resource management system that runs on a Unix/Linux platform. The system administrator is responsible for maintaining the Unix operating system, the Oracle Utilities Network Management System, and the PC connections to remote workstations. This guide provides details about installing, optimizing, and troubleshooting the Oracle Utilities Network Management System and assumes that the reader is an experienced Unix/Linux user.

System Overview

An Oracle Utilities Network Management System includes:

- Isis
- Oracle Utilities Network Management System services
- Oracle Utilities Network Management applications
- User environments
- A tablespace in an Oracle database

The Oracle Utilities Network Management System can be broken down into individual components. Each component is installed and configured separately. Oracle Utilities Network Management System uses a client/server architecture. The server supports Oracle Utilities Network Management System daemon processes, while the clients display a graphical user interface to allow the user to interact with the system. Internal daemon service process to daemon service process communication is managed with a concurrency management and messaging system called Isis. Isis is the backbone of the communication architecture for an Oracle Utilities Network Management System. The network model, system configuration, and operational data is all stored persistently in an Oracle database.

The table below describes the Oracle Utilities Network Management System components.

Component	Description
Client User Environments	The Java-based end-user environments are configured using a combination of SQL files (RDBMS table based configuration), XML files, and Java properties files. The XML files are based on an NMS-specific XML schema, which provides the foundation for Java user interface customization.
Isis	Clients access services and tools through a central concurrency management and messaging system called Isis. Isis is a real-time implementation of message oriented middleware that helps provide access to the Oracle Utilities Network Management System daemon service processes as well as inter-daemon process communication.
Services	Services maintain and manage the real-time electrical network data model. Services also cache information from the database tables to optimize client information access.
Applications	Applications consist primarily of the front-end tools used by the Operational User. These tools access data from the services for presentation to the user and perform specific actions corresponding to appropriate business practices.
Web-Gateway	The Web-Gateway is a combination of a CORBA (Common Object Request Broker Architecture) interface and Oracle WebLogic Application Server. The Web-Gateway allows messages published via Oracle Utilities Network Management System Services to be made available to Java (Swing) clients. The Web-Gateway also provides a mechanism for the Java clients to request information on or request updates to the Oracle Utilities Network Management System run-time model.
Oracle Database	The Oracle Database contains the complete network data model, configuration, and operational data history of an Oracle Utilities Network Management System.

Note: Services, applications, and the Oracle RDBMS tablespaces can be spread over multiple servers or run on a single server. The simplest configuration is for everything (Oracle RDBMS, Oracle Utilities Network Management System services and Oracle WebLogic Java Application Server to run on a single (generally SMP) server. Common variations would include the use of a cluster based hardware server to support high-availability (for Oracle RDBMS and Oracle Utilities Network Management System Services). This provides flexibility for system configuration, depending on your needs and hardware.

User Environments

Oracle Utilities Network Management System forms a managed visual workspace that organizes tools into related groups allowing users to perform specific tasks. Each group of tools makes up a separate user environment, or user type. The user type, entered when logging in to the application, establishes the environment by specifying the scripts that are run to launch tools.

The Java/Swing-based end-user environments are configured using a combination of sql files (RDBMS table based configuration), XML files, and Java properties files. The XML files are based on an Oracle Utilities Network Management System XML schema that allow the Java user interface to be customized for a particular project implementation.

Isis

Isis is the common messaging bus through which client processes and services interact on a Unix TCP/IP network. Isis is primarily concerned with passing messages between Unix processes on the network.

There are multiple hardware and message bus configurations that can be applied within the scope of a single Oracle Utilities Network Management System. The most common configuration is to run Isis on a single server (or node of a Unix cluster). Isis is a real-time synchronous/asynchronous real-time pub-sub bus. Processes subscribe to updates that are published by other daemon processes on the bus.

Database

Oracle Utilities Network Management System requires an Oracle relational database management system (RDBMS). The database persistently manages the tables that define the information constructs of the electrical network data model (sometimes called an operations model). Oracle Utilities Network Management System services cache information from the relational tables. These tables include the management of system constructs such as handles and aliases, class hierarchy, topology model, device status, events, incidents (trouble calls), outages, and conditions.

Database installation and configuration follow these basic steps:

- The Oracle RDBMS is initially configured using the product's standard installation and configuration procedures. To help you get started, Oracle provides an example network data and customer model (Oracle Power and Light) that can be installed out of the box.
- Using Oracle Utilities Network Management System utilities, the initial schema is installed and populated. For a new project installation (not out of the box Oracle Power and Light), note that significant work must generally be undertaken to translate available electrical network topology and customer model data into the standard schema required by the Oracle Utilties Network Management System. This is an effort often measured in months, not days. Proper conversion of available network and customer data to the standard Oracle Utilities Network Management System schema is generally the most time consuming aspect of a project implementation.
- If you are performing an upgrade, you may need to perform a migration of the schema and population of the database.

All Oracle Utilities Network Management System schema definitions follow the SQL standard. Schema installation and population use SQL scripts that are generally executed via the SQL interface (ISQL.ces) to the Oracle RDBMS instance. The necessary data elements required for an Oracle Utilities Network Management System consist of the following components.

Component	Description
Oracle Tablespaces	Used for persistent storage of production data (e.g., network components, operations data, etc), customer information and indexes. The Oracle Utilities Network Management System model is typically loaded into three or more separate tablespaces, Electrical Network Operations data, Electrical Network Operations index data and Customer Model data (name, address, phone, account, etc.).

Component	Description
Maps	Maps are collections of model element data (mostly coordinates) typically grouped by electrical feeder but sometimes grouped by geographic area. These maps are used to minimize RDBMS access and increase performance during graphical map rendering. These maps are stored in the $OPERATIONS_MODEL directory (usually $NMS_HOME/data). There two types of maps: electrical and background. Electrical maps can always be regenerated from the RDBMS. Depending on how background maps are built for your model, background maps may not be. Some models build background maps as translations of background data from the customer-specific master GIS. Maps are tied to a specific database and cannot be associated with any other.

Hardware and Third Party Software

Since specific system requirements can change with new releases, they are not available as part of this document. For the most current requirements, refer to the *Oracle Utilities Network Management System Quick Reference Guide.* Information on de-supported platforms and deprecated integrations is provided in the *Oracle Utilities Network Management System Release Notes.*

Network Architecture

Running Oracle Utilities Network Management System software over a shared local area network and wide area network requires a network analysis. Network latency can cause significant problems with an Oracle Utilities Network Management System. Since significant inter-process communication is managed by Isis via TCP/IP, significant latency or constrained network bandwidth can cause slowdowns, reduced throughput and possibly process shutdowns.

Security Guidelines

The Oracle Utilities Network Management System (NMS) utilizes several Unix ports to facilitate communication between various daemon processes. Some of these ports are used to communicate to adapters to external (non-Oracle) systems. Below is a list of the common ports NMS might utilize in a production environment. It includes a brief description of what each port (or set of ports) might be used for, whether these ports are configurable, and whether these ports would expect to be exposed on a production server (to facilitate communication to external systems).

> **Note**: the most secure Isis configuration is standalone mode. In this "mode," only processes executed on the same physical server as Isis can access the bus at all - and all inter-process communication is over the localhost (loopback) address - minimizing external exposure. This is the configuration specified in the Isis parameter file template. Standalone mode is set to true by setting "isis_standalone 1" value in the $ISIS_PARAMETERS Isis configuration file.

Port	Configurable?	Internal?	User	Description
2042	Y	Y	ISIS	ISISPORT – connection to Isis.

Port	Configurable?	Internal?	User	Description
2043	Y	N	ISIS	ISISREMOTE – remote connection to Isis.
49152->65535	N	Y	ISIS	Transient TCP/UDP ports used for Isis process to process communication.

Architecture Guidelines

This chapter provides an overview of the product module dependencies and locations, the logical hardware relationships, and sample physical hardware implementations:

- **Product dependencies and locations**
- **Logical hardware design**
- **Sample server implementations**
- **Hardware sizing**
- **Printing**

Overview

The guidelines in this section complement the information contained in the Oracle Utilities Network Management System Release Notes. The Product Summary and Dependencies document has been replaced by a combination of the Release Notes and this Architecture Guidelines document.

This section contains information about product module dependencies and locations, the logical hardware relationships, and sample physical hardware implementations. It should provide the information needed to understand the relationships between the software modules and the hardware that is required to implement.

For an overall product summary, please refer to the Oracle Utilities Network Management System User Guide.

Product Dependencies and Locations

The following table describes Oracle Utilities Network Management System product module dependencies and their locations.

Module/ Component	Product	Dependency	Server	Client	Location
Model Management	OMS Base / DMS Base				
NMS Core Services			Unix		System Server

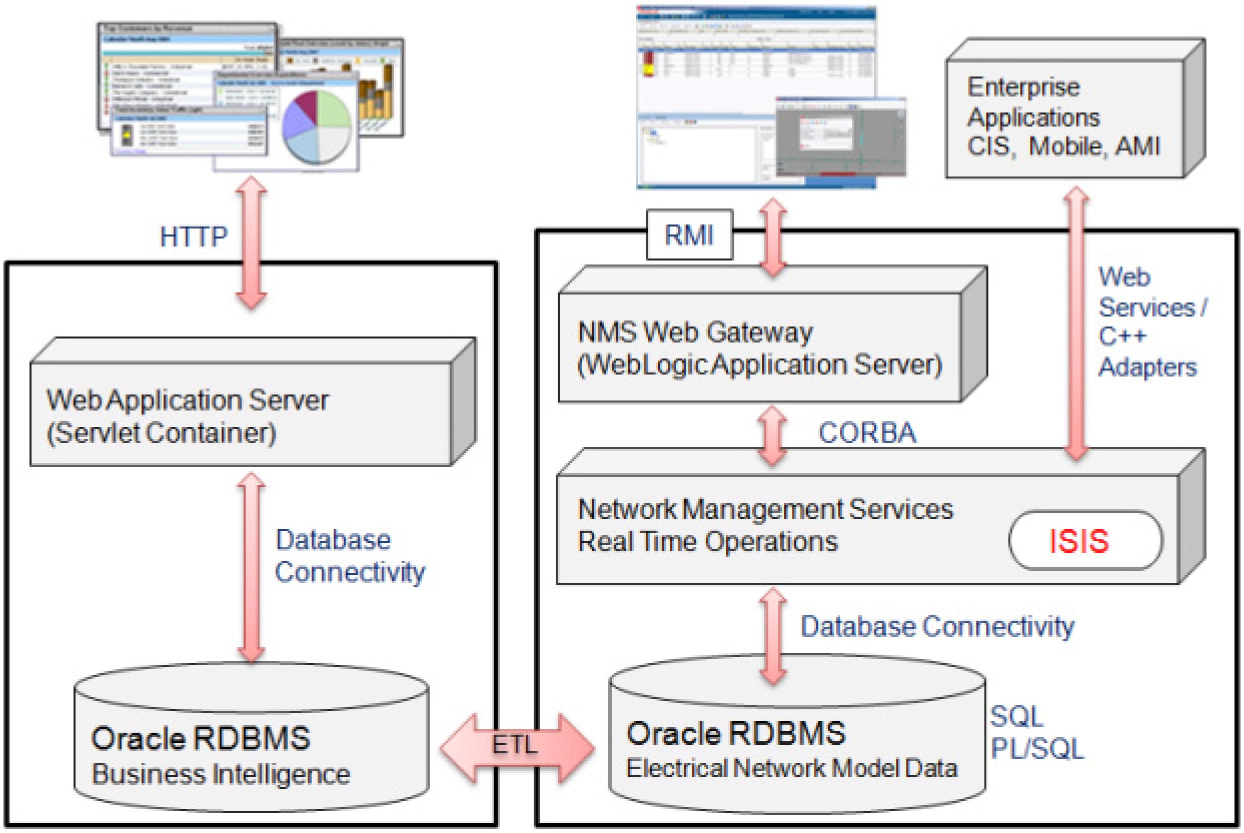
Module/ Component	Product	Dependency	Server	Client	Location
Web Gateway			Unix		System Server
Configuration Assistant				Windows	Web Client
US Electric Ops Model	OMS Base / DMS Base	Model Management			
Model Builder			Unix		System Server
US Standard Configuration	OMS Base / DMS Base	Model Management			
Web Trouble	OMS Base	Web Workspace			
High Availability	OMS Base / DMS Base	Model Management			
Cluster Capability			Unix		RDBMS Server/ System Server
Redliner	OMS Base / DMS Base				
Redliner Application				Windows	Clients
GIS Adapters	OMS Base / DMS Base	Model Management			
ESRI Adapter					GIS Server
Intergraph Adapter					GIS Server
Smallworld Adapter					GIS Server
Generic Adapters	OMS Base				
IVR Adapter		Web Trouble	Unix		System Server

Module/ Component	Product	Dependency	Server	Client	Location
CIS Adapter		Web Trouble	Unix		System Server
Switching Management	OMS Base / DMS Base	Model Management			
Switching Service			Unix		System Server
Switching Application				Windows	Web Client
Power Flow Extensions	DMS Power Flow	OMS Base or DMS Base			
Power Flow Service		Model Management	Unix		System Server
Power Flow Applications		Web Workspace		Unix	Application Server
Suggested Switching	DMS Adv. Feeder Mgmt	Power Flow Extensions	Unix		System Server
Feeder Load Management	DMS Adv. Feeder Mgmt	Power Flow Extensions	Unix		System Server
Fault Location, Isolation & Service Restoration	DMS FLISR	Switching Management and SCADA Adapters	Unix		System Server
Volt/VAR Optimization	DMS VVO	Power Flow Extensions	Unix		System Server
Fault Location Analysis	DMS FLA	Power Flow Extensions	Unix		System Server
Schematics	OMS Switching & Schematics / DMS Base	Model Management			
Schematics Generator			Unix		System Server
Generic MQ Adapters	OMS Adapters				
CIS MQ Adapter		Web Trouble	Unix		System Server

Module/ Component	Product	Dependency	Server	Client	Location
CIS MQ Callback Adapter		Web Trouble	Unix		System Server
IVR MQ Adapter		Web Trouble	Unix		System Server
Mobile MQ Adapter		Web Trouble	Unix		System Server
Generic Adapters	OMS Adapters				
AMR Adapter		Web Trouble	Unix		System Server
Storm Management	OMS Storm	Web Trouble		Windows	Web Client
Web Workspace	OMS Base DMS Base	Model Management		Windows	Web Client
Web Trouble	OMS Base	Web Workspace		Windows	Web Client
Web Call Entry	OMS Call Center	Web Trouble		Windows	Web Client
Web Callbacks	OMS Call Center	Web Trouble		Windows	Web Client
Call Overflow Adapter	OMS Call Center	Web Trouble	Unix		System Server
SCADA Extensions	NMS SCADA	Web Workspace		Unix	Application Server
SCADA Adapters	NMS SCADA				
ICCP Blocks 1 & 2			Unix		ICCP Server
ICCP Block 5			Unix		ICCP Server
Generic SCADA			Unix		System Server
Service Alert	OMS Paging				
Service AlertService		Web Trouble	Unix		System Server
Service Alert Client		Web Trouble		Windows	Web Client

Module/ Component	Product	Dependency	Server	Client	Location
NMS Schema	NMS Extractors & Schema	Model Management	Unix		BI RDBMS Server
Outage Analytics	Schema	NMS Extractors and Schema and Web Trouble		Windows	BI Web/ App Server
Distribution Analytics	Schema	NMS Extractors and Schema and Power Flow Extensions		Windows	BI Web/ App Server
Trouble Reporting	NMS Extractors and Schema	Web Trouble		Windows	BI RDBMS Server BI Report Server
Storm Reporting	NMS Extractors and Schema	Storm Management		Windows	BI RDBMS Server
Switching Reporting	NMS Extractors and Schema	Switching Management		Windows	BI RDBMS Server

Oracle Utilities Network Management System High-Level Conceptual Diagram



Hardware Sizing

Hardware sizing guidelines are not discussed in this document. There are many variables that affect hardware sizing and the calculations would be more complex than what is suitable for this document. Hardware sizing is best handled by the Consulting Services team that is working on the project.

Chapter 2

Standard Product Implementation

This chapter provides an overview of a standard implementation of Oracle Utilities Network Management System, including:

- **Overview**
- **Software Release Level**
- **Installation**
- **Interfaces**
- **Modeling and GIS Integration**
- **Operations Modules Software Configuration**
- **Management Reporting Modules Software Configuration**

Overview

Changes to the Oracle Utilities Network Management System standard product software, installation, interfaces, modeling, and software configuration are considered typical and acceptable for a standard product implementation. Staying within the guidelines discussed in this guide allows a customer to follow the standard configuration from release to release and significantly reduces Oracle Utilities Network Management System migration and upgrade issues.

The intent is to allow a customer to make changes that follow the 80/20 rule; that is, a customer should be able to stick to 80% of the standard product configuration and only make the 20% configuration changes which are absolutely necessary to make the implementation successful.

There are many additional configuration changes possible and technically supported by Oracle; however, changes outside of these guidelines are considered project scope changes and redefine the project as a non-standard configuration project. This in turn creates testability and maintainability issues, as non-standard configuration may not be encompassed by our test process and can result in issues with which our customer support department may not be familiar. In addition, deviations from the product configuration mean your system will not conform as closely to standard product documentation and training material.

Software Release Level

A standard product implementation should utilize a release of Oracle Utilities Network Management System with no software code changes, additions or modifications. The software should be on an officially supported release code line and not a special project code line. Only patches that are produced by the Oracle support organization and/or the project team should be installed when necessary to fix critical problems.

Installation

The installation should be done according to the guidelines taught in the Oracle Utilities Network Management System System Administration class and follow all recommended procedures for system configuration. The software should be installed on servers and clients in a configuration that meets the requirements stated in the Architecture Guidelines section of Chapter 1 for the installed modules. The installation also should comply with the required operating system level and patches identified in the Oracle Utilities Network Management System Installation Guide. The utilized Oracle Utilities Network Management System software modules should have all dependent Oracle Utilities Network Management System software modules installed and configured. The required third-party products should be installed and at the supported release level as stated in the Oracle Utilities Network Management System Installation Guide document for the installed release.

Interfaces

A standard product implementation should use the Oracle Utilities Network Management System standard CIS, IVR, and mobile data interfaces with an officially supported middleware gateway such as the WebSphere MQ gateway or using the Oracle Table Interface. SCADA system integration should be done utilizing the Oracle Utilities Network Management System LiveData ICCP Adapter, Generic SCADA, or MultiSpeak-based web services SCADA adapter. AMR/AMI and AVL integrations should be done using the Oracle Utilities Network Management System MultiSpeak Adapter. Paging and email notification integration should be done using the Service Alert supported services.

When interfaces are done to non-standard systems that cannot be supported utilizing the standard interfaces described above, they should be done utilizing the published APIs and should not directly read or write to the Oracle Utilities Network Management System operations database.

Database level and/or reporting integration may be done using the Oracle Business Intelligence for Utilities database and must utilize tables and attributes described in the Oracle published schema.

Modeling and GIS Integration

The following sections describe some recommended guidelines to follow when you integrate Oracle Utilities Network Management System with a GIS.

GIS Model Extractor

The GIS extractor utilized should either be supported by Oracle or by one of our modeling partners. The extractor should produce Oracle standard model preprocessor (MP) files and utilize the Oracle conventions for model building and an approved incremental update process.

Standard Preprocessor

The Oracle Utilities Network Management System standard preprocessor supports eighteen different rules that allow for data translation (for instance, expand elbows, or add recloser bypass switch). It is acceptable to use as many of these rules as necessary to build an acceptable operations model. The standard preprocessor takes as input model preprocessor (MP) files and produces Oracle standard model build (MB) files.

Device Types and Attributes

Select which device types (classes) are used from the standard model definition, mapping the customer's GIS data to these existing classes.

- Define unique class alias names based on the GIS attribute(s).
- Select which attributes are used from the standard model definition (providing at a minimum those necessary for the required modules), again mapping the customer's GIS data to the existing attributes.
- Utilize Oracle-provided modeling workbooks to define the model used for the project, which is used to generate the project classes and inheritance.
- The name of any device may be constructed from one or more GIS attributes.
- The display name for any device type can be changed (for instance, allows the device type to have a different name on the control tool).

Software Configuration Dependencies On Device Types

There are a number of NMS software configuration aspects that depend upon the device types that are chosen to be built within the NMS data model. In so far as the data model can change for different facilities, the software configuration must be adapted. The following configuration settings are dependent upon the resulting NMS model definition and require adaptation for every project. These configurations are generated automatically by Oracle to match the defined NMS model.

- Control Tool panels
- Web Trouble Stop Classes
- Symbology mapping and symbol set

Operations Modules Software Configuration

This section lists configuration options in Oracle Utilities Network Management System applications and components, including:

- **Web Workspace**
- **Web Trouble**
- **Web Call Entry**
- **Web Callbacks**
- **Web Switching Management**
- **Power Flow Extensions**
- **Fault Location Analysis (FLA)**
- **Fault Location, Isolation, and Service Restoration (FLISR)**
- **Feeder Load Management (FLM)**
- **Suggested Switching**
- **Volt/VAr Optimization**
- **Redliner**
- **SCADA Extensions**
- **Service Alert**
- **Storm Management**

Overview

Unless there is sound reason to change them, Oracle recommends that labels, buttons, table columns and dialogs be left as-is for consistency. This avoids confusion and further improves our ability to support our customers. However, there are cases where such changes are allowed, and the following sections identify those cases. There are also cases where it is allowable to delete a field, button or label. This may mean that the deleted item is actually just "hidden". Depending upon where on the form the deleted or hidden item was originally placed, there may be some "white space" remaining where the deleted item was present.

Web Workspace

Login

- Add and remove usernames (using the Configuration Assistant).
- Delete or rename user types.

Work Agenda

- Change labels of any column.
- Change labels of any menu/toolbar items.
- Add three permanent filters (using the Configuration Assistant).
- Add three permanent sorts.
- Change set of Work Queues (or Dispatch Groups).

Main Menus/Toolbar

- Delete or rename items.

Authority

- Define specific control zone hierarchy (up to 5 levels).

Viewer

- Change project symbology file used by Oracle Utilities Network Management System (Customer responsibility - includes AVL crew symbology if configured.
- Viewer background color may be gray or black.
- Annotation and/or landbase color may be changed to be compatible with the Viewer background. All annotation is assumed to be one color, and all landbase graphics are assumed to be a single color.
- Zoom levels.
- Declutter / reclutter.
- Big Symbols.
- Selectable and unselectable objects.

Control Tool

- Change labels for actions.
- Delete actions.

Web Trouble

Event Management Rules

- Delete any standard rule set.
- Change parameter values of any rule in any standard rule set (using the Configuration Assistant).
- Delete any rule in any standard rule set (except in cases where there are rule dependencies).

Event Details

- Delete outage reporting drop down menus.
- Rename outage reporting drop down menus.
- Add and delete items on outage reporting drop down menus (using the Configuration Assistant).
- Add additional option menu field verification prior to completion (e.g., not only must the Failure and Remedy be changed from “Unselected”, but it may also check for values in other option menu fields prior to completion).
- Remove current completion validation check or any other configured validation check.

Crew Actions

- Add and Remove Crew Types from standard list of crew types.
- Add and Remove Personnel Job Titles from standard list of job titles.
- Add and Remove Vehicle/Equipment types from standard list of vehicle/equipment type.
- Add and Remove Control Zone filter interactions with the Work Agenda.

Damage Assessment

- Add, remove, or rename damage types.
- Modify the minutes to repair, and minutes to repair if inaccessible, for each damage type.
- Add, remove, or rename damage parts.

Web Call Entry

- Add and remove usernames - using the Configuration Assistant.
- Add/Remove Trouble Codes but must map to Oracle standard trouble codes .
- Change labels and order of columns in Outages Summary.
- Modify Event History Cause dropdowns to reflect outage reporting drop down menus.

Web Callbacks

- Add and remove usernames - using the Configuration Assistant.
- Add/Remove Callback Status options but must map to Oracle standard callback statuses.
- Change labels of any column in main window and View My Callback Lists window.

Web Switching Management

Switching List/Safety List

- Change labels of columns.
- Change labels of menu/toolbar items.

Switching Documents

- Change labels on any header field.
- Delete any header field.
- Change header labels on any switching step field.
- Add additional required fields verification prior to state change.
- Remove any validation check or any other configured validation check.

Safety Documents

- Rename any safety document.
- Change labels on field of standard documents.
- Add additional required fields verification prior to state change.
- Remove any validation check or any other configured validation check.
- Delete any fields of standard documents.
- Delete any standard documents.
- Define up to three new safety documents (starting from a copy of any standard safety documents and making any of the allowable changes listed above).

Power Flow Extensions

Power Flow User Tools

- Change labels of columns on Power Flow Results.
- Remove columns to display on Power Flow Results.

Power Flow Algorithm Rules

- Change parameter values of any power flow algorithm rule - using the Configuration Assistant.

Load Profile

- Number of day types.

Seasonal Conductor and Transformer Flow Ratings

- Seasonal limit.
- Normal limit.
- Emergency limit.

Power Flow Switching Extensions

- Change labels of Power Flow specific columns on switching steps.

Fault Location Analysis (FLA)

- Change labels of any column.
- Change ordering of columns.
- Change formatted string value for “Distance from Upstream Switch” column, for example from ft to yds or meters, depending on the GIS units used.

Fault Location, Isolation, and Service Restoration (FLISR)

- Change labels of any column.
- Change labels of any button.

Feeder Load Management (FLM)

- Change labels of any column.
- Change ordering of columns.

Suggested Switching

- Change labels in Suggested Switching user tools.

Volt/VAr Optimization

- Change labels of on any screen.

Redliner

There are no configuration options available.

SCADA Extensions

- Change labels of columns on SCADA Summary page.
- Change tooltips of buttons on SCADA Work Agenda page.
- Change alarm limit values.

Service Alert

Service Alert provides a user interface for update and maintenance of the contact list, notification parameters, customer contact information, and critical/sensitive customer information; it is the customer's responsibility to do this administration via the provided tool. You may modify XSL messages for use by the supported notification mechanisms/devices.

Storm Management

- Change labels of columns.
- Change labels of menu/toolbar items.
- Change the historical average lookup values, but not how they are used in the algorithm.
- Define a sort order for the events that is used by the analysis engine prior to stepping through its periodic analysis, within the constraints of the configuration options available for this purpose.
- Change storm outage type names, definitions and restoration order, within the constraints of the configuration options available for this purpose, including adding or removing some (but not all) outage types (directly tied to the historical average lookup value definition process).
- Specify which of the top three control zone levels is the “simulation level” (the level at which the lookup values are specified).
- Define which crew types are eligible to assess/repair which storm outage types.
- Define performance factors for each crew type.
- Define nominal crew resources.
- Change storm shift definitions, within the constraints that there must be at least one but no more than four shifts, and the sum of all shift lengths must be exactly 24 hours (directly tied to the historical average lookup value definition process).
- Change storm season definitions, within the constraints that there must be at least one but no more than four seasons, and each month must be part of a season (directly tied to the historical average lookup value definition process).
- Change storm holiday definitions, including the removal of all holiday definitions (directly tied to the historical average lookup value definition process).
- Change storm special conditions types (directly tied to the historical average lookup value definition process).
- Change storm level names, including adding or removing some (but not all) levels.
- Change list of company names for the crew resources, including adding or removing some (but not all) names.
- Add and remove usernames and passwords.
- Delete or rename user types.

Management Reporting Modules Software Configuration

The following sections describe the cases where it is allowable to change values in the Management Reporting modules.

Business Intelligence

Allowable values you may change include:

- Modify extractor to match configuration within guidelines of accepted product configuration changes.
- Oracle provides standard Oracle Business Intelligence for Utilities dashboards and answers; it is the customer's responsibility to modify these to meet business needs.

Trouble Reports

Allowable values you may change include:

- Oracle provides standard reports in Oracle BI Publisher or Oracle BI Discover (customer choice); it is the customer's responsibility to modify these to meet business needs.

Chapter 3

Unix Configuration

Oracle Utilities Network Management System is installed and configured on a Unix or Linux workstation or server. The workstation or server must be properly configured before running the software. This chapter describes the Unix configuration required for optimal use of Oracle Utilities Network Management System. It includes the following topics:

- **Unix User Names**
- **Korn Shell**
- **Executables/Run-Times**
- **Operating System Configuration**

Unix User Names

Oracle requires creating a Unix user in order to administrate the Oracle Utilities Network Management System.

Creating an Administrative User

The administrative user, as the name implies, has central control over many critical aspects of the Oracle Utilities Network Management System. This user is the central controller of:

- Isis – configuration and starting and stopping of the Isis processes
- Oracle Utilities Network Management System services – Stopping and starting and repository of service logs
- Oracle Utilities Network Management System binaries – compiled code, configuration files.
- Database connection that has write privileges as well as read privileges
- Model-building data.

It should be noted that for data security, Oracle Utilities Network Management System tools that can be used to directly modify data are installed with permissions set so that only the administrative user is allowed to execute them.

The administrative user (e.g., nms) has access to critical components of the system. This user owns and maintains the services, the starting of the services, model building, binaries, the database, and the configuration standards. The administrative user maintains the Oracle Utilities Network Management System Unix-based configuration and executables in one location. The administrative user is configured with an Oracle Wallet and the environment variable $RDBMS_HOST, which point to the ORACLE production tablespace. Thus, when services are started the user has the necessary read/write access to the production tablespace.

The administrative user:

- Owns the executable and runtime directories.
- Has read-write permissions to the production database.
- Owns the service processes (DBService, MTService, etc.)
- Performs all sms_start.ces commands.
- Performs all model builds.

Korn Shell

The Korn Shell sets environment variables and provides a command line interface to the operating system. The Korn Shell (ksh) standardizes command line execution and requests, such as running scripts, executing applications, and operating the services. The Korn Shell uses a file called .profile to configure itself. Both the administrative and application users need to have

- Their default shell set to ksh.
- The .profile configured to source the Oracle Utilities Network Management System configuration file (.nmsrc).

For your convenience, templates of a generic .profile and .nmsrc file are included in the Oracle Utilities Network Management System software distribution, under $CES_HOME/templates. These files can be copied to $NMS_HOME/.profile and $NMS_HOME/.nmsrc and then modified to suit your installation.

.profile Configuration

The Korn Shell is configured using .profile file. It is a hidden file that exists in the user's home directory. When a user logs in, this file executes, setting environment variables and defining terminal configuration. The following is required for setting up .profile.

The .profile file must source the user environment configuration file, .nmsrc. This is an easy addition to .profile. Add the following line to the bottom of .profile using any text editor.

```
. ~/.nmsrc
```

This runs .nmsrc in the current shell and initializes all of the environment variables within the .nmsrc file in the current working environment.

The .profile file must also execute correctly when called from another script, as well as when the user logs in at a terminal. Anything in .profile that is terminal-specific should be placed in an "if" clause to suppress execution if the .profile is not being run from a terminal.

```
# Set a variable to be true when .profile is
# being run from a terminal rather than a script.
#
if tty -s
then
TTY=true;
else
TTY=false;
fi
#
# Protect items that must only be run from a
# terminal and not from a script.
#
if $TTY
then
stty Compaq
tset -I -Q
```

```
PS1="`hostname`>"
fi
```

The search path environment variable, $PATH, tells the operating system where to locate the files necessary for software execution. It must include the directories that contain the Oracle Utilities Network Management System Unix-based software. The entry in .profile will look something like the following example, where *<project>* is the application user name:

```
export PATH=/users/<project>/bin:/users/nms/bin:$PATH
```

This entry searches the user's home directory before the nms directory, letting customer specific tools and scripts take precedence over the Oracle Utilities Network Management System base executables provided.

Executables/Run-Times

The Oracle Utilities Network Management System Unix-based software is installed in the product home directory ($CES_HOME/bin). When commands are entered at the prompt, the shell looks for the appropriate bin directory for a matching program. The PATH environment variable determines where the shell looks for the bin directory, so PATH must be modified to include the location of the Oracle Utilities Network Management System software. It is defined in the .nmsrc file located in the user's home directory and it may contain multiple path names, each separated with a colon (:). The shell parses each path name until the corresponding program is located or each path name is exhausted.

> **WARNING!** The .nmsrc file sets up the PATH environment variable to ensure that the correct executables are discovered in the correct order. If you need to modify the PATH environment variable, it should be done in the .profile, after the .nmsrc is run, and you should only append directories to the end of the list. Doing otherwise could cause problems with your system.

Operating System Configuration

A standard operating system installation will often not be optimally configured to work with an Oracle Utilities Network Management System. Sometimes the user will spawn more processes than allowed by the standard kernel configuration. Other times, a map file may require a larger data segment than the average user. Due to problems like these, you may find that you will have to tweak the operating system configuration, which may include reconfiguring the kernel or some other part of your Unix system.

The values that are specified in this guide are examples only, as the correct values depend on how large your operating model is, how you use the system (e.g., as a server, app-server, or client) and what kind of a load is placed on the system. This section should give you an idea of how to change components of the operating system that frequently become a problem running Oracle Utilities Network Management System.

Solaris

In Solaris, limits to data segment size and the number of files available to the user are defined by the ulimit command. For the most part, these parameters do not need to be tweaked, but should you need to, you can run:

```
$ ulimit -d <datasegment size in kilobytes>
```

(Usually 256 Mb will be enough)

```
$ ulimit -n <number of file descriptors>
```

(Usually 1024 will be enough)

AIX

AIX sets its limits in a system configuration file called /etc/security/limits. You can type “man limits” from the command line for the documentation on how to modify this file. The following table describes the parameters you may have to modify.

Parameter Name	Description
Nofiles	The soft limit on the number of open file descriptors
nofiles	The hard (upper) limit on the number of open file descriptors
data	The soft limit on data segment size
data_hard	The hard (upper) limit on data segment size

Users can adjust these parameters using the ulimit command, as long as the parameters are below the hard limit configuration. If your parameter requirements are under the hard limit, you may want to consider adding the appropriate ulimit command to the .nmsrc file instead of modifying the limits file. For example, adding a line that states ulimit -d 262144 would set the data segment size limit to 256 MB; having it in the .nmsrc file would ensure that the limit is set correctly each time the user logs in.

In addition, AIX supports a range of “network options” that may need to be tuned for optimal performance. Specifically, it is generally recommended to set the following AIX network options (via root on the AIX server).

```
$ no   -p -o rfc1323=1
$ no   -p -o sb_max=2097152
$ no   -p -o tcp_sendspace=524288
$ no   -p -o tcp_recvspace=262144
$ no   -p -o udp_sendspace=65536
$ no   -p -o udp_recvspace=655360
$ no   -r -o ipqmaxlen=512
```

Linux

In Linux, limits to data segment size and the number of files available to the user are defined by the ulimit command. For the most part, these parameters do not need to be tweaked, but should you need to, you can run:

```
$ ulimit -d <datasegment size in kilobytes>
```

(Usually 256 MB will be enough)

```
$ ulimit -n <number of file descriptors>
```

(Usually 1024 will be enough)

Core File Naming Configuration

Unix systems can generally be set up to save a core file if an executable experiences a non-recoverable error of some sort. Standard Unix configuration generally names this file "core" and places it in the directory where the executable was executed. The problem with this configuration is that if a core file does get generated it can happen that a second core file (from the same or different executable) can overwrite the original core file - thus hiding information that could possibly be used to better track down the source of the problem. This is not an entirely uncommon phenomenon for Unix system and there are generally Unix OS specific steps that can be taken to have the OS generate core files with process specific names - to help prevent information from being lost and make it easier to solve problems if they do occur. Below are some OS specific options for this purpose:

Solaris

As root edit /etc/coreadm.conf (COREADM_INIT_PATTERN=core.%p) or run coreadm -i "core.%p".

AIX

From your .nmsrc file:

```
export CORE_NAMING=true
```

or - as the root user - if you want to change it for the entire system. Do man on chcore for more info.

```
chcore -n on -d
```

Linux

Note the following may be the default on the Linux distributions supported by Oracle Utilities Network Management System.

```
echo "1" > /proc/sys/kernel/core_uses_pid
```

You should also check to make sure the ulimit for core files is set to unlimited - otherwise no core or a truncated core file may be created:

```
ulimit -c unlimited
```

Chapter 4

Isis Configuration

Isis is the backbone of the Oracle Utilities Network Management System. It is the messaging bus through which all components communicate. This chapter provides the details for configuring Isis. It includes the following topics:

- **Isis Configuration Files**
- **Isis Architecture**
- **Isis Directory Structure**
- **Isis Environment Variables**
- **Isis Log Files**
- **Starting Isis**
- **The cmd Tool**
- **Troubleshooting**

Isis Terminology

The following table describes Isis terms used in this chapter.

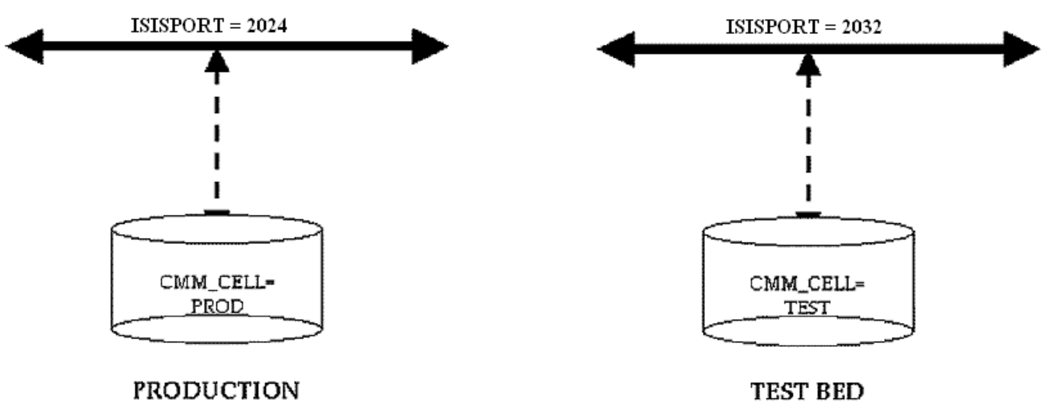
Term	Definition
Ports	Isis requires a set of three TCP/IP ports for communication. These are defined in the sites file. These ports may also be defined in the /etc/services file. The port definitions in the /etc/services file may be overridden through the use of the ISISPORT and ISISREMOTE environment variables. ISISPORT defines the UDP port that Isis backbone sites use to communicate with each other and the TCP port that processes use to connect to the Isis backbone. Thus ISISPORT defines two out of the three TCP/IP ports for running Isis. Default value for ISISPORT is 2042, as registered by the Internet Assigned Numbers Authority (IANA). ISISREMOTE defines the UDP port used by processes to communicate to the Isis backbone when no TCP ports are available. Specifying this field is necessary even thought it is generally not used by Oracle Utilities Network Management System. Default value for ISISREMOTE is 2043, as registered with IANA. If ISISREMOTE is NOT provided it will default to $ISISPORT+1 via the isisboot startup script.

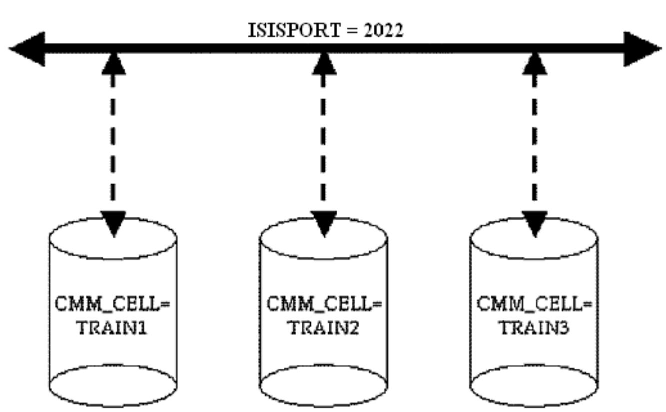
Site	Each node (client workstation or server) that runs the Isis protocol is a site. Each site has a defined number (1, 2, 3...) and a set of TCP/IP ports that are used to communicate with it, as defined in the sites file. The set of ports assigned to each site are generally the same for each site.
protos	Protos is the name of the Isis protocol process. Each Isis backbone site has one copy of this process.

Isis Architecture

The following diagrams show an architecture in which a production system is running on a machine(s) (Services and Applications) with a specified ISISPORT, while additional Oracle Utilities Network Management System applications are run on separate ISISPORTs. The Isis process we are most concerned with is isis-protos - often referred to as simply protos. To have an operational Isis messaging backbone you must have access to at least one protos on your network (normally the same machine you are running on). Separate Oracle Utilities Network Management System applications can be run using the same ISISPORTs as long as the CMM_CELL names are unique. However, it is highly recommended that a production system retain its own individual (private) port. Sharing a protos process between production and non-production Oracle Utilities Network Management System environments is not recommended.

The diagrams below show a production system on its own port and CELL, a test bed with its own port and CELL, and a training system with a single port supporting multiple CELLs for individual training environments.





Stopping an Isis protos process associated to a port (*e.g.*, 2042) stops all Oracle Utilities Network Management System services and applications in every CMM_CELL (e.g., PROD) associated with that Isis port (2042). This does not affect any Isis processes running on other ports (2032, 2020).

Stopping services and tools within a CELL does not affect the Oracle Utilities Network Management System services and tools in any other CELL.

Isis Directory Structure

The Isis directory structure is provided for verification purposes only.

Directory	Contents
bin	Isis executables, including the isisboot script, which is used to start up Isis. The cmd command resides here as well. cmd provides a command line interface to Isis that is useful for verifying connections and debugging problems.
lib	Isis runtime libraries
include	Contains Isis include files used in compiling the software

run_isis

The run_isis directory should normally be under $NMS_HOME/etc/ and contains Isis configuration files:

- sites, which defines all of the nodes on a given Isis "backbone"
- isis.rc, which provides startup information for Isis
- isis.prm, which is the Isis parameter file. The location of the Isis parameter file can also be overridden via the ISIS_PARAMETERS environment variable.

Isis Configuration Files

This section addresses the files that affect the configuration of Isis software. Some of these files are Isis specific files, while others are operating system files.

sites File

The Isis sites file is located in $NMS_HOME/etc/run_isis. It identifies all nodes on the network that will be running Isis, assigns them a unique Isis identification/site number, and defines the TCP/IP port numbers under which they will run.

The standard Isis sites file should always specify the standard ports you see in the example below - for all sites involved. If you wish to override the standard Isis ports set the ISISPORT environment variable accordingly (see notes below on ISISPORT). The isisboot script will examine the ISISPORT environment variable and - if set to anything other than 2042 - will create an alternate sites file (sites.$ISISPORT) with the proper port configuration.

This file must be updated and consistent across all nodes running Isis in the computer network. For each entry in the sites file, a corresponding entry should exist in the /etc/hosts file in case the DNS services fail. The format of this file specifies the Isis node number, network service ports, hostname, user name, and a comment:

```
+ 001:2042,2042,2043 server1.oracle.com ces,hp9000s800
+ 002:2042,2042,2043 server2.oracle.com ces,hp9000s800
+ 003:2042,2042,2043 client1.oracle.com ces,Alphaserver
+ 004:2042,2042,2043 client2.oracle.com ces,Alphaserver
+ 005:2042,2042,2043 client3.oracle.com ces,hp9000s700
+ 006:2042,2042,2043 client4.oracle.com ces,E450
+ 007:2042,2042,2043 client5.oracle.com ces,E250
```

The leading plus sign is very important and this file cannot have any comments (except in the comment section of the end of each line).

isis.rc Startup File

The isis.rc file is located in $NMS_HOME/etc/run_isis. It contains the following information:

- The Isis processes to start
- The location of Isis logs

A generic isis.rc license file is now included in the Oracle Utilities Network Management Systems software distribution. Previously each customer was provided with a customized isis.rc file, but this is no longer a requirement and the isis.rc provided with the distribution should be used instead.

/etc/hosts

The /etc/hosts file is a Unix operating system file. It defines all of the nodes in the computer network configuration. The format of this file specifies the Internet Protocol address, hostname and any aliases, all separated by tabs. Comments begin with a #. Also, there should be an alias provided for all machines, which is less than 15 characters, for the Isis processes; display hosts managed by the applications cannot have more than 15 characters.

```
# See the hosts (4) manual page for more information.
# Note: The entries cannot be preceded by a space.
#  The format described in this file is the correct format.
#  The original Berkeley manual page contains an error in
#  the format description.
127.00.0.1localhostloopbackloghost
200.100.100.1 server1.oracle.comserver1
200.100.100.2 server2.oracle.comserver2
200.100.100.3 client1.oracle.comclient1
200.100.100.4 client2.oracle.comclient2
200.100.100.5 client3.oracle.comclient3
200.100.100.6 client4.oracle.comclient4
200.100.100.7 client5.oracle.comclient5
```

Isis Environment Variables

Isis environment variables let client user names connect to different user environments. For example, a user might switch from a configuration environment to a model build environment.

ISISPORT and ISISREMOTE

ISISPORT is set to the second (tcp) service port in the sites file, and ISISREMOTE is set to the third (bcast) service port in the sites file. Note the second (tcp) port must be the same as the first (udp) port. These variables tell the tools and services where to listen for Isis messages. If ISISPORT is not defined, it defaults to 2042 (via the isisboot script). If ISISREMOTE is not defined, it defaults to $ISISPORT+1. To simplify configuration it is generally recommended to NOT specify ISISREMOTE (let isisboot take care of it for you).

ISISHOST

This variable specifies a possible list of nodes on which Isis will be running. It is used to configure a remote Isis system. A remote Isis system is one in which Isis and some applications run on different nodes. The applications on startup will cycle through the comma delimited list specified by this variable and will seek to make a UDP connection with the Isis process running on the remote node until it finds a valid Isis process with the same ISISPORT and ISISREMOTE values. This value can be for fail over; the applications will check each node in the list on startup, and if the first is down, they will connect with the next in the list. This is not needed or used with most implementations of Oracle Utilities Network Management System.

CMM_CELL

This variable lets different sets of services and tools exist on the same service ports. Messages received on the ports from tools and services started with a different CMM_CELL are disregarded.

CMM_CELL can be set to any value, as long as it is unique from other CMM_CELL variables running on the same service ports.

ISIS_PARAMETERS

Specifies the Isis parameter file to be referenced by applications and services on startup.

An example Isis parameter file is provided in the templates directory (isis.prm.template). Unless there is a specific reason not to, it is suggested that the default template be used for production systems. By default (if no parameter file exists) the isisboot script should automatically put in place and use a copy of the default Isis parameter template file. Existing customers should verify that their Isis parameter file and the provided template are in reasonable agreement (there should be a rational explanation for differences). An example of possible parameter file content for an Oracle Utilities Network Management System appears below:

```
#isis.prm
isis_NativeThreadStackSize 131072
# specify that all applications should provide their
# parameters when a dump occurs
isis_prmDumpAllParameters 1
# allow messages which can have 10MB of information, model
# builds may require messages of this size
isis_msgMessageSizeLimit 10000000
#
isis_UDPSndbuf             131072
isis_UDPRcvbuf             131072
#
isis_iclPacketHighWaterMark 49152
```

```
isis_iclPacketLowWaterMark 32768
# don't go below 2048
isis_iclMaxSlots           4096
#PROTOS
protos_maxLocalClients 1024
protos_maxRemoteClients 1024
protos_taskHigh 100
protos_taskLow 95
```

Isis Standalone Mode

By default, Isis now starts in "standalone" mode. This means that Isis will bind to the local loopback adapter (localhost - 127.0.0.1) and not the adapter defined by the gethostbyname function. This means that Isis will not be available to other hosts on the network by default, and this is generally desireable. If your configuration requires a connection to Isis from another host, you will need to edit the Isis parameters file ($NMS_HOME/etc/run_isis/isis.prm) and change "isis_standalone" from "1" to "0":

```
isis_standalone  0
```

Disabling Isis on Network Adapters

If you are not running Isis in standalone mode, Isis will bind to all available network adapters on the server. It is good practice (and sometimes necessary) to configure Isis to not bind to certain adapters. An example would be a heart-beat network that is typically configured on a clustered server. To disable an adapter, list it's IP address or subnet in the Isis parameters file ($NMS_HOME/etc/run_isis/isis.prm or whatever file is pointed to by the $ISIS_PARAMETERS environment variable):

```
isis_rnsDisable_1   192.168.123.0/24
isis_rnsDisable_2   10.10.42.20
```

See the isis.prm.template file in the templates directory for further documentation.

Isis Multi-Environment Considerations

When configuring multiple Oracle Utilities Network Management System environments, each site should be assigned a unique port number ($ISISPORT). This logically partitions each network and prevents unwanted cross effects. As an example, the on-line system environment may be assigned to the 204x ports while the model build environment may be set to 214x, the off-line engineering environment set to 224x, and so on. While it is possible to configure all systems on the same port with CMM_CELL values differentiating the systems, it is not recommended.

Isis Log Files

isis.<date>.<time>.log

The isis.<date>.<time>.log file keeps track of events while Isis is initializing and is essentially the output file for the isisboot script. It is located in $CES_LOG_DIR (or under $ISIS_LOG_DIR if it is set). The isis.<date>.<time>.log file contains clues if there is any difficulty in starting Isis.

<Site No.>.logdir

This is the directory where the Protos and Incarnation logs reside. The location of this directory is defined in the isis.rc file, and is typically found in $CES_LOG_DIR/run.isis (or under $ISIS_LOG_DIR/run.isis, if $ISIS_LOG_DIR is defined).

The Protos Log

protos is the Isis protocol process. This process logs its messages to $CES_HOME/etc/run_isis/<Site No.>.logdir.<port>/<Site No.>_protos.YYYYMMDD.HHMMSS.log. Check here for runtime problems with Isis. Each time Isis is restarted, the previous protos log is moved to the old_log subdirectory. The retention period for logs in the old_log directory is driven by the $CES_DAYS_TO_LOG environment variable or for 7 days, if the environment variable is not set. Each time Isis is restarted it will delete logs older than the retention period.

The Incarn Log

This is a short file called $CES_LOG_DIR/run.isis/<Site No.>.logdir./<Site No.>.incarn and it usually includes a single line containing the incarnation number for the particular site.

Starting Isis

isisboot

isisboot is the script that initializes Isis. On startup isisboot reads the isis.rc license file to determine if it can proceed. If so, it reads the sites file to determine the default network ports and site (node) identification numbers to use.

Initializing Isis

To initialize Isis, complete these steps:

1. From the nmsadmin user name type:

   ```
   isisboot
   ```

2. When complete (which could take up to a minute or more), type:

   ```
   cmd status
   ```

   This determines if Isis has successfully started and will provide information similar to the following:

   ```
   cmd:  my_site_no = 1
   my_host = 127.0.0.1
   Isis version = V3.4.14 Build: 20 $Date: 2010/06/09 19:03:03 $
   verbose mode = off
   ```

3. If it has started successfully, type:

   ```
   cmd sites
   ```

   Result: Isis lists all connected machines. For example:

   ```
   tstaix01:cesadmin$ cmd sites
    ***  viewid = 1/1
     tstaix01.oracle.com             [site_no 1 site_incarn 3]
   ```

Starting Isis on Non-Default Ports

Isis may need to run on ports other than the default ports listed in the sites file. It is common to separate different sets of services by running Isis for those services on separate network ports. For example, a configuration system may run on 1601, 1602 and 1603, while the model build services run on ports 1701, 1702 and 1702. Therefore it may be necessary to switch a client from one set of Isis ports to another.

To start Isis on a non-default port, complete these steps:

1. To check which ports to use, at the prompt type:

   ```
   echo $ISISPORT
   ```

   This returns the port configured for this environment.

2. As the nmsadmin user, set ISISPORT to the desired Isis port number. For example:

   ```
   export ISISPORT=2032
   ```

3. Then run isisboot as per usual. The isisboot script will take care of setting up Isis on the proper ports.

   Results:

   - A new sites file called sites.$ISISPORT will be created from the existing sites file.

The cmd Tool

Verify the connection to Isis using the cmd tool. If cmd is working, Isis is functioning as well. The syntax for cmd is:

```
$cmd <options>
```

Type cmd from the Unix command line to bring up the command line interface, identified by the `cmd>` prompt. The following table presents a subset of cmd commands.

Command	Description
sites	Shows all nodes connected by Isis on the current ports: cmd>sites *** viewid = 34/5 test1.oracle.com [site_no 34 site_incarn 1] test2.oracle.com [site_no 33 site_incarn 1] test3.oracle.com [site_no 6 site_incarn 1]
status	Provides the current status of the Isis protos process. Part of the information returned is the current Isis version corresponding to the executed cmd binary.
list	Provides a list of all the Isis process groups and applications connected to the protos. This identifies the CMM_CELL and process group. This can be used to identify remaining processes still running.

Command	Description
snapshot	Sends a message to all applications currently connected to Isis to generate an Isis dump. All the Isis related information for this process is written to disk in a log file with the process ID as the prefix (<pid>.log). This log file can be found in the run.*Service directories for services or the directory from which applications have been launched. Isis dumps are extremely useful when debugging problems, as they can tell the developer exactly what messages are being processed at the time the dump was generated.
rescan	Tells protos to update the site view.
shutdown	Causes the protocols process to shutdown. Wait 4 minutes before restarting Isis after a shutdown or an unsuccessful start attempt, and verify that all processes are completely down by checking the process list on each node (ps -aef).
Help	Print all cmd command options.
Help <command>	Print information about a specific command.

Exiting cmd

Type “quit” to exit cmd.

Troubleshooting

When an Oracle Utilities Network Management System application or Service is experiencing problems, some helpful information would include an Isis dump of the applications process, the log file associated with that application, and the output of the processes list.

Generating an Isis Dump File

To generate an Isis dump file, complete these steps:

1. On each node of concern:

```
ps -aef > $(hostname)_ps.out
```

2. Identify the process ID of the problem application(s).

```
grep -i <application> $(hostname)_ps.out
```

3. Use the following command to generate an Isis dump file for a specific process:

```
kill -USR2 <pid>
```

The process will not be affected and will continue to operate, but upon receiving the USR2 signal, it will generate an Isis dump (<pid>.log) in the directory from which that tool was launched.

> **Note:** Any subsequent USR2 messages will result in the process appending a new Isis dump to the <pid>.log file. Only the user running the applications can perform this action.

Generating an Isis Dump File for All Applications

An alternative is to issue the cmd snapshot command, which will create an Isis dump for all applications. The applications will continue to run, but every single application running will create an Isis dump file. This will clutter the file system, but it is sometimes the best way to gather all the information you need to investigate a problem.

To issue the cmd snapshot and obtain a list of all the current Isis dump files, enter these commands:

```
cd $NMS_HOME
ps -aef > $(hostname)_ps.out
touch DUMP_START
cmd snapshot
find . -name [1-9]*.log -newer $NMS_HOME/DUMP_START > ~
       /logs.txt 2>/dev/null
echo $(hostname)_ps.out >> ~/logs.txt
zip isis-dumps.zip ‘cat ~/logs.txt‘
```

This set of commands will find all the .log files that start with a number and were generated after the time the DUMP_START time-stamped file was created. It will create a file called `isis-dumps.zip` that can be sent back to Customer Support for investigation. With the full set of logs, Customer Support can track all the interactive messaging for problem investigation and resolution.

Reporting a Problem to Customer Support

In general, when reporting a problem to Customer Support, the following information can speed the problem identification and resolution process:

- An explanation of what the observed symptoms were and where they occurred.
- An explanation of how to repeat the problem, if possible.
- An explanation of expected behavior.
- A specific time frame when the problem occurred.
- Example data demonstrating the problem (e.g., event numbers, crew names, etc.).
- Service logs and environment log files of the affected Services/Applications.
- Isis dumps of the affected application and services at the time the problem was observed. A complete Isis dump of all processes may be requested if the problem is repeatable, along with a process list output file.
- The core file trace, if a core file exists for the process.
- Any other activity that occurred prior to, or concurrent with, the issue that may stand out as a possible contributor.

Chapter 5

Database Configuration

Oracle Utilities Network Management System currently supports the Oracle Relational Database Management System (RDBMS). The RDBMS must be properly installed and configured prior to using the Oracle Utilities Network Management System software. This chapter provides the configuration requirements for Oracle. It includes the following topics:

- **Oracle Installation Guidelines**
- **Oracle Tablespaces**
- **Oracle Users**
- **Starting Oracle**

Oracle Installation Guidelines

It is recommended that Oracle Enterprise Edition be installed. Please see the Oracle RDBMS installation documentation for specific Oracle installation requirements.

Oracle Tablespaces

Every Oracle Utilities Network Management System must have its own Oracle tablespace set. In general, the tablespaces consist of the following:

Tablespace	Description
Production	The production tablespace (ces_db) contains all of the production data for Oracle Utilities Network Management System. This includes model data, outages, and data that is produced by operations performed in Oracle Utilities Network Management System.
Production Temporary (Optional)	The production temporary tablespace (ces_tmp) temporarily stores operating data prior to insertion into the production tablespace. The default Oracle TEMP tablespace should be the designated temporary tablespace for the system. Oracle is more efficient when managing temporary data in this way. Make sure that a sufficient amount of space is allotted to TEMP.
Production Index	The production index tablespace (ces_idx) contains all of the indexes for the production tablespace. The nms user's .nmsrc file must contain the CES_INDEX_TABLESPACE environment variable referencing this tablespace.

Tablespace	Description
Customer Data	The customer data tablespace (*<project>*_customers_db) belongs to the customer. It is populated with the entire customer database by the CIS extraction process. Public synonyms are assigned to the customer tables and selectability is granted to production Oracle users so that the necessary table joins can be created.

Each tablespace should be located on a separate disk to enhance performance and decrease bottlenecks due to high volumes of input/output.

It is key that the tablespaces are provided with sufficient disk space and are monitored regularly for growth. When a tablespace runs out of disk space, operational data will be lost and Oracle Utilities Network Management System services will discontinue to function properly.

Oracle Instances

For performance, scalability and simplicity there is normally only one Oracle instance on a production machine. It is not generally recommended that a production machine have multiple Oracle instances on the same machine. An exception would be where a cluster is used; you may want an Oracle instance installed on both sides of the cluster (production on the primary side, Model Build, Test, or Oracle Business Intelligence on the secondary side). Under normal circumstances there would only be one instance of Oracle on each side – if one side of the cluster fails you could end up with two instances on the surviving node. In general, try to keep it simple.

You should consult with your Oracle Utilities Network Management System Professional Services technical team to develop a creative solution to meet your specific needs.

Oracle Utilities Network Management System uses an Oracle Wallet and the environment variable RDBMS_HOST to create a connection to the Oracle database. The wallet stores the database user and password in encrypted form, and the RDBMS_HOST (which is the tnsname for the database instance) points the Oracle client software to the correct user/password pair within the wallet. The wallet and related environment variables are created and maintained by running the script config_nmsrc.pl as the NMS administrative user.

Each instance of Oracle Utilities Network Management Systems (*e.g.*, production, test, model build) must have a unique database owner with its own tablespaces. Using the same database owner for two implementations will result in corrupted data.

> **Note:** Two or more Oracle Utilities Network Management System instances on a single machine can be acceptable (depending on machine resources) for testing, training and model build environments.

It may be necessary to tune Oracle for the specific environment it will be operating on. Typically a qualified DBA can perform the necessary tuning and modifications. Often this is an iterative process that requires running the full Oracle Utilities Network Management System on the production machines and capturing statistics for analysis.

Other Environment Variables

Other Oracle-specific environment variables may need to be different between systems, but these are due to how the DBA has constructed the environments. Other than the NLS specific environment variables noted below, these are listed in one of the example tables in chapter five.

When Oracle is loaded onto a given platform, the Oracle instance itself will generally have a default National Language Support (NLS) setting. Oracle Utilities Network Management System client applications (like DBService) which utilize the Oracle Call Interface (OCI) need to know what NLS settings to use for inserting and interpreting result sets from Oracle. Presently, the easiest way to do this is as follows:

1. Add the following environment variable to your .nmsrc file: NLS_LANG

   Note: For a US configuration, Oracle believes the NLS_LANG environment variable (as far as OCI is concerned) typically defaults to AMERICAN_AMERICA.WE8ISO8859P1. Thus if a customer sets their Oracle NLS to something other than this value (inside of Oracle during instance setup) - and does not specify the NLS_LANG environment variable to appropriately match, DBService will not start. You will see a note in the DBService log file indicating a mismatch that must be rectified.

2. The following process should work for setting NLS_LANG:

   ```
   Set NLS_LANG to
   <NLS_DATE_LANGUAGE>_<NLS_TERRITORY>.<NLS_CHARACTERSET>
   ```

   where each “NLS component” needs to match the values returned by this query:

   ```
   SELECT * FROM v$nls_parameters WHERE parameter IN
     ( 'NLS_DATE_LANGUAGE',

   'NLS_CHARACTERSET', 'NLS_TERRITORY' )
   ```

   For example, we have NLS_LANG=AMERICAN_AMERICA.WE8ISO8859P1, and our query returns:

   ```
   PARAMETERVALUE
   -----------------------------
   NLS_DATE_LANGUAGEAMERICAN
   NLS_TERRITORYAMERICA
   NLS_CHARACTERSETWE8ISO8859P1
   ```

3. Set the ORA_NLS10 environment variable. For example:

   ```
   export ORA_NLS10=/users/oracle/product/10/nls/data
   ```

   (or wherever your valid Oracle nls/data directory is located)

Oracle Users

Once the tablespace is established, you must create users and grant their permissions. Oracle users are those users that have access to the Oracle tablespaces. Before defining the users, it is important to discuss the security role that a user can possess.

Security Roles

Security roles determine the level of database operations that a user can perform. There are two types of security roles:

Role	Description
ces_rw	Read-write role. This role has read and write privileges to the production data. It can create, drop, update to, and insert to, all of the production tablespace objects.
ces_ro	Read-only role. This role can only connect and select data from the production tablespace objects. **Note:** Certain security tables, such as ces_users, are excluded from the view of the ces_ro role.

Users

There are three Oracle users. Each user directly relates to the tablespaces. Substitute specific customer name for <project> where noted below.

User	Description
<project>_CES	The <project>_ces Oracle user is the owner of the production tablespace. This user has a ces_rw role and maintains full control of the data elements in the production tablespace.
<project>	The <project> Oracle user is the application user. This user has a ces_ro role to the production tablespace.
<project>_customers	The <project>_customers user has full privileges to the customer data tablespace only and no privileges on the production tablespace.

Starting Oracle

Complete the following steps to start Oracle:

1. Login as oracle. If logged in as the root user, the system will not request a password. At the prompt, type:

   ```
   su - oracle
   ```

2. Login to SQL*Plus:

   - As the oracle user, type:

     ```
     sqlplus /nolog
     ```

   - At the SQL> prompt, type:

     ```
     connect / as sysdba
     startup
     quit
     ```

3. Start the listener. As the oracle user, type:

   ```
   lsnrctl start
   ```

Note: The tnsnames.ora and listener.ora files must be properly configured to start the oracle listener. The location of these files may vary by system, but they must be consistent on all machines requiring connections via SQLNET.

4. Login as the distribution user and test the connection to Oracle. At the prompt, type:

```
ISQL.ces
```

This references the RDBMS_USER, RDBMS_PASSWD and RDBMS_HOST to establish the connection to the database. If this connection is successful, a SQL> prompt will appear.

Chapter 6

Environment Configuration

This chapter includes the following topics:

- **Encrypting Configuration Parameters**
- **The System Resource File**
- **Modifying Environment Variables**

Encrypting Configuration Parameters

Some environment and configuration parameters contain sensitive information, such as authentication credentials, that should be protected. This section provides two methods for encrypting cleartext strings.

Encrypting Passwords with Oracle WebLogic Server Utility

Passwords stored in the ces_parameters table and the CentricityServer.properties, MWMInterface.properties, and AMRInterface.properties files can be stored in encrypted form. These passwords are encrypted/decrypted using an encryption key that is unique to each WebLogic domain. To generate an encrypted password, run the following commands as the WebLogic user:

```
$ cd <domain-dir>/bin
$ . ./setDomainEnv.sh
$ java weblogic.security.Encrypt
```

This will prompt for the password and then output the encrypted version of that password, which can then be copied to the appropriate SQL or properties file.

Generating key.client.pass with the Client Keystore Password Utility

The CentricityTool.properties key.client.pass property is used to access the public key in the nms-client.keystore. The Client Keystore Password utility (*client-keystore-password*) encryptes the key.client.pass property.

To generate the encrypted property, run the utility as the NMS admin user:

```
$ client-keystore-password
```

The utility will prompt for the password and then write it to the key.client.pass property in $NMS_CONFIG/jconfig/global/properties/CentricityTool.properties.

To specify a different **.properties** file:

```
$ client-keystore-password -f filename.properties
```

The System Resource File

The System Resource file ($HOME/.nmsrc) houses the environment variables that enable the Oracle Utilities Network Management System to operate correctly and consistently. They define the connections information for the database and Isis, as well as environment specific configuration settings, such as viewer symbology.

You will need to modify the System Resource file in part for application to specific systems. One suggestion is to use environment variable dependencies. By doing this you can simplify the process of changing values; by changing one variable that is a root dependency, the change will cascade through a number of others, limiting your required changes and maintaining consistency throughout the file.

Modifying Environment Variables

Because of the innate flexibility allowed by environment variables, there are an infinite number of permutations you can apply for a system setup. Not everything you can do with these variables should be done. This section describes the suggested settings that you should adhere to in order to avoid confusion.

To modify the environment variables, complete these steps:

- Modify the variable you want to change with the new settings in the .nmsrc file.
    - Type **.nmsrc** at the prompt to source the file in the current working environment. The new variables replace the old variables.

    Note: New variables replace the old variables when the file is sourced. You should source .nmsrc each time you change the file. The file .profile automatically sources .nmsrc at startup.

Environment Variables

The table below lists the required environment variables and their standard settings that must be modified depending on the type and number of environments you are constructing. See templates/nmsrc.template for more variables. Other variables may be added as well, depending on the functionality of your system. This is not an exhaustive list, but it does address the variables typically required to start an Oracle Utilities Network Management System.

Environment Variable	Example Setting	Description
NMS_ROOT	/users/nmsadmin	Provides a common location to place the base Oracle Utilities Network Management System directories and files owned by the administrator. It is recommended that you set this to the home directory of the Oracle Utilities Network Management System administrator. By specifying this directory correctly, you can use it to simplify other installations. When this value changes, the change will be cascaded throughout the other dependent environment variables. This environment variable is used by a number of scripts and processes.
CES_HOME	$NMS_ROOT/nms/product/1.10.0.0	This environment variable is set to the product installation directory for the active installation.

Environment Variable	Example Setting	Description
NMS_HOME	${HOME}	The nmsadmin username home directory. This is the directory where the implementation directory and runtime directories exist. This should be set to the nmsadmin username home directory.
NMS_CONFIG	$NMS_HOME/ $NMS_PROJECT	This is the location of the project configuration and implementation files. The name (*e.g.*, OPAL) must also match the CES_SITE variable on the left side (*e.g.*, "OPAL product ces").
CES_DATA_FILES	$NMS_HOME/sql	Defines the location of data files used in various scripts and routines that define aspects of system configuration. This variable must be defined and can be accessed from a number of scripts. The standard location for these files is the $NMS_HOME/sql directory. Examples are ces_classes.dat, ces_inheritance.dat, and ces_devices.cel.
CES_DAYS_TO_LOG	5	Identifies how long to store the old log files. When services are restarted, log files older than the set number of days (5 in this case) will be removed.
CES_DATA_TABLESPACE	ces_db	Used by the ISQL.ces process to identify the tablespace name of data tablespace.
CES_INDEX_TABLESPACE	ces_idx	Used by the ISQL.ces process. It will parse SQL scripts that create indexes and make sure that the index is actually created in the specified tablespace name. This tablespace must be owned by the user configured in the Oracle Wallet. The practice of separating indexes from operational data improves Oracle performance.
CES_LOG_DIR	$HOME/log	For services and login environments, this defines where to place the resulting log files. Since log file generation requires write access for a process, the user who started the process must have write access to this directory. It is highly recommended that this directory be located on a different filesystem from $CES_HOME.
CES_SITE	Project specific. Example: <project> product ces	Defines the configuration inheritance path for a system. When the setup process executes, it searches this site variable from left to right looking for configuration files with prefixes that match the value in the site variable. This feature lets you inherit or override the ces or product configuration. This variable is used by most of the configuration scripts.
CES_SYSDATE	Environment specific. Example: %D%R|%D|%R	Defines the display format for which all applications will display date and time elements. The format for this requires specifying 3 formats: date and time | date | time The three formats specified in this environment variable must also be added to the $DATEMSK file.

Environment Variable	Example Setting	Description
CES_SMTP_SERVER	smtp.example.com	The hostname or IP address of a Simple Mail Transfer Protocol (SMTP) server. This is used by ServiceAlert when sending alert emails. See also: CES_DOMAIN_SUFFIX.
CES_DOMAIN_SUFFIX	example.com	The domain suffix to be used when sending emails via ServiceAlert.
CMM_CELL	Environment specific. Will be specified to some unique value for each system. Example: production	Allows for encapsulation of Isis messages within a specific group of the same CMM_CELL specification. All applications that join up and connect to a specific set of services must have the same CMM_CELL, as well as ISISPORT and ISISREMOTE variables.
DATEMSK	$NMS_HOME/etc/ ces_datefmt	This file will be generated and updated by Oracle. It defines all the expected date formats that can be encountered as input by widgets and Services. Services will use the values in this file, for example, as a format dictionary when given call time as part of a trouble call. Expected time formats should be placed near the top of the file so that the search and compare algorithm encounters the most likely values as quickly as possible.
ISIS_PARAMETERS	$NMS_HOME/etc/ run_isis/isis.prm	Identifies which file to reference for Isis parameters. This must be established before initiating an application.
ISISPORT	System specific, the default should be 2042.	A TCP/IP connection port on which Oracle Utilities Network Management System processes communicate (via Isis).
ISISREMOTE	System specific, the default should be 2043.	A TCP/IP port used when you are making a connection to a “remote” protos. This can either be when the process is running on a machine without protos or if a local connection is attempted and all the local connections are filled.
NLS_LANG	System specific. Example: AMERICAN_AMERICA.WE8MSWIN1252	The National Language Support value for the Oracle database installation. DBService will not start unless this is set correctly. To definitively determine what the various NLS_LANG components should be for your RDBMS instance, the following query should be helpful: select * from v$nls_parameters where parameter in ('NLS_DATE_LANGUAGE', 'NLS_TERRITORY', 'NLS_CHARACTERSET')
NMS_APPSERVER_HOST	System specific. Example: server.example.com	The hostname of the Java application server. Needed for sites running WebLogic

Environment Variable	Example Setting	Description
NMS_APPSERVER_PORT	System specific: Example: 7001	The port on which the Java application server at NMS_APPSERVER_HOST is listening. The WebLogic default port is 7001.
NMS_NS_HOST	System specific. Example: server.example.com	The hostname where the Naming_Service is started. Only needed for sites running a Java application server.
NMS_NS_PORT	System specific. Example: 17821	The port on which the Naming Service is running. Only needed for sites running a Java application server.
OPERATIONS_RDBMS	System specific. Example: ces_db	Identifies the primary tablespace for the operations data.
ORACLE_HOME	System specific: Example:/usr/users/oracle/ product/11	Identifies the home directory for the Oracle user. This is necessary to simplify other variables dependant on this path.
ORACLE_SID	System specific. Example: PRODSERV01	Identifies the Oracle session ID value.
PREFERRED_ALIAS	Model specific. Example: OPS:PSU	Defines what alias of a device is to be displayed by default. In the example, the system will display the alias that has a DB_TYPE of OPS as found in the alias_mapping table. If an alias with a DB_TYPE of OPS does not exist, then the PSU (pseudo) alias will be displayed. This, by convention, is a unique name of <class_name.device_idx>. Depending on the model build definition, you can use and define other alias options, such as a SCADA alias.
RDBMS_HOST	System specific. Example: PRODSERV01.world	Identifies the host machine for establishing an sqlnet connection via Oracle. This value must exist in the tnsnames.ora file on the system attempting a connection. This is required for the use of many setup scripts and ISQL.ces.
ORACLE_SERVICE_NAME	System specific	The service name of the Oracle database that the system should connect to.
SYMBOLOGY_SET	System specific. Example: $OPERATIONS_MODELS /SYMBOLS/ PRODUCT_SYMBOLS.sym	Identifies the primary symbology file loaded by the Viewer. This file identifies the Viewer symbols for all objects.
CES_HAS_NMA	1	Indicates the NMAdapter is active for this environment. The NMAdapter can be used to support certain interactions between NMS and the Java Application Server.

Environment Variable	Example Setting	Description
NMS_WEB_USER	WebGw	Username used to identify the source of communication from the NMAdapter to the Java Application Server.
NMS_WEB_PASSWD	passwd	Password used to authenticate communication into Java Application Server from the NMSAdapter. See **Encrypting Configuration Parameters** on page 6-1 for details on encrypting configuration parameters/ passwords.

Chapter 7

Services Configuration

The configuration of Oracle Utilities Network Management System services involves establishing the location of system services on server nodes in the computer network and defining their configuration and command line options.

This chapter includes the following topics:

- **Services Overview**
- **Service Alert Email Administration**
- **Service Alert Printing Administration**
- **Services Configuration File**
- **Model Build System Data File**
- **Starting and Stopping Services**

Services Overview

Oracle Utilities Network Management System services provide memory-based model management for RDBMS persistent electrical network model information - generally to support real-time access and performance objectives. The services maintain the memory resident data model for the real-time status of the electrical network. The memory model caches the necessary data to build the model from relational database tables. The services then solve this model (fills in the blanks, determine what is energized, grounded, looped, etc.) and optimize the result for client access. Each service generates and passes appropriate incremental model updates to Isis (the Network Management System real-time publish/subscribe message bus) for publication. Interested applications subscribe to the published messages keep the Network Management System end users up to date with current state of the model.

Startup scripts that run when the operating system boots can be used to automatically start the Oracle RDBMS, Isis, and Oracle Utilities Network Management System services. How you configure and where you place these scripts is based upon startup (default) Unix "runlevel" and your platform. For Linux platforms you can generally determine your current runlevel via:

```
/sbin/runlevel
```

For Linux startup/shutdown scripts are generally located in the /etc/init.d directory. A Unix softlink to each startup script to run for a given runlevel is generally made in the /etc/rc<run_level>.d directory. Other Unix operating systems have similar but often slightly different conventions. It is presently an exercise left to the system administrator to properly create and configure startup scripts that will properly run on startup for a given Operating System. Example scripts for some common startup scripts may be found in the $CES_HOME/templates directory. These scripts are examples only and will need to modified/reviewed/tested locally to ensure they work properly for a given installation.

Oracle Utilities Network Management System Services are generally flexible and attempt to cater to the functional needs of various utility clients through the use of command line options and run-time parameters stored in the relational database. Below is a brief summary of the primary Oracle Utilities Network Management System Services. Details about available command line options and relational database parameters specific to each service can be found in the $CES_HOME/documentation/services directory.

SMService - System Monitor Service

SMService monitors the core Oracle Utilities Network Management System service and interface processes. It reads and caches the appropriate system.dat configuration file to determine which processes to initiate and monitor. In the event that a managed process fails, SMService restarts it based on the cached configuration data from the system.dat file.

The following variations of system.dat files should be located under $NMS_HOME/etc. There should be *.template versions of these files in the $CES_HOME/templates directory. These configuration files generally define the specific run-time executables and command line options necessary for a given Network Management System installation:

- system.dat.init - defines configuration required for initial setup.
- system.dat.model_build - defines minimum configuration required for initial model builds.
- system.dat file - defines configuration for fully operational Network Management System.

sms_start.ces will launch SMService, which in turn will cache the $NMS_HOME/etc/system.dat file by default and then launch the remaining services, interfaces and adapters as defined by the $NMS_HOME/etc/system.dat file. The following command sequence can be used to specify an alternate system.dat type file:

```
sms_start.ces -f ~/etc/my_system.dat
```

The smsReport tool can be used to request and monitor the SMService view of the processes it is currently managing. smsReport is a non-GUI tool used to report the state of the system by querying SMService. It is executed in either one-shot or monitor mode. One-shot mode is the default mode that queries SMService for the current state and displays it to the user on exit. However, if the system state is INITIALIZING, then smsReport automatically switches to monitor mode so as to not exit prior to initialization completing before exiting. Monitor mode is set by starting smsReport with the -monitor command line option. It serves the same function as the default one-shot mode but does not close after the system state has been reported.

To shutdown the Oracle Utilities Network Management System (gracefully) use the following script:

```
sms_stop.ces
```

The sms_stop.ces script will shutdown all of the user environments (one at a time) and then the services in reverse order to how they where defined to startup in the ~/etc/system.dat file. Using this script generally prevents certain deadlock conditions which can occur if an attempt is made to stop all user environments and system services at the same time.

DBService - Database Service

DBService provides database access for any processes attached directly to the Isis message bus within the Oracle Utilities Network Manage System environment. The messaging backbone, Isis, directs database queries and commands to the appropriate Oracle RDBMS server and returns results to the requesting process.

> **Note**: A given instance of DBService allows a configurable number of queries to occur in parallel but serializes RDBMS updates. By assigning update responsibility of specific tables to specific DBService instances (by convention) parallel updates can be supported which generally increases performance and/or scalability under system load. TCDBService, MBDBService are examples of this strategy.

ODService - Object Directory Service

ODService registers new objects as well as caches configuration and (optionally) run-time information that is likely to be requested by applications in a particular form and/or on a regular basis. This caching allows the requests to be handled very quickly without directly accessing the database. Cached information is primarily static configuration data, such as object classes, class hierarchy, symbology assignments and (optionally) device alias information.

DDService - Dynamic Data Service

DDService manages real time (dynamic) information required by the system. In addition to command line options DDService utilizes the srs_rules table for run time options.

Examples of dynamic data that DDService manages include:

- Current status of switchable devices
- Special operating conditions of devices (tags, crews, notes, etc.)
- SCADA information (analogs, digitals, quality codes)
- Operating authority (users and control zones)

When you make changes to Oracle Utilities Network Management System control zones (control_zones and/or control_zone_structures tables), you need to tell DDService to update its internal control zone memory structures with the following UpdateDDS command:

```
UpdateDDS -recacheZones
```

When you make changes to SCADA device definitions, you can tell DDService to update itself with the following UpdateDDS command:

```
UpdateDDS -recacheMeasures
```

MTService - Managed Topology Service

Real-time electrical systems are in a constant state of flux of electrical flow. A single device operation could de-energize a model section, create a parallel on one or more phases, ground one or more phases, create a loop condition, or extend some other form of energization/deenergization. Since the topological state (*i.e.*, energization, ground status, energizing feeder, feeder color, etc.) of a device cannot be accurately determined without taking into account a large number of other devices and operating conditions, it is not possible for each application to independently determine current topological states. Instead, MTService maintains a complete topological copy of the model in memory, which it updates as devices and conditions change. It publishes topological impact updates and services topological data requests from other Network Management System applications and services.

JMService - Job Management Service

JMService is the customer trouble call analysis engine. It relies on MTService to trace device connectivity when determining probable outages in the system. Customer complaints (trouble calls) are fed into the system and JMService groups them using configurable rules to compute and publish the most likely cause of the problem. JMService also manages restoration resources (crews). In addition to command line arguments, JMService uses the srs_rules table for the majority of its run-time configuration options.

TCDBService - Trouble Call Database Service

This is a copy of DBService that runs specifically to improve the performance of JMService by handling database calls for JMService. This lets the main DBService manage database requests from operator activity not directly related to trouble calls.

MBService - Model Build Service

MBService is used in building a data model, which mirrors the customer's existing data model (generally extracted from a Geographic Information System such as ESRI, Intergraph, SmallWorld, or AutoCAD). When changes are made in the GIS a project-specific extractor is used to extract and transform GIS changes into a standard Network Management System format. MBService takes the standardized input, parses and integrates the resulting changes into the Oracle Utilities Network Management System electrical network model. In addition to maintaining the model database, MBService also generates map files, which are optimized for use with Network Management System graphical viewing tools.

SwService - Switching Service

SwService helps manage switch plan state transitions and provides a facility for sending updated plans to interested parties via e-mail.

MBDBService - Model Build Database Service

MBDBService serves the same purpose for MBService as TCDBService does for JMService. It is a copy of DBService that runs specifically to improve the performance of MBService by handling the database calls resulting from model building. It only applies if you use the -mbdbs command line option when starting MBService. This option bypasses DBService and uses MBDBService to perform queries and SQL commands.

MQDBService - MQService Gateway DBService

MQDBService provides direct access to the database for the MQSeries Gateway. This reduces competing throughput for the DBService reserved for operator interactions.

PFService - Power Flow Service

PFService manages real-time operations power flow calculations that allow you to view the complex voltages and currents at points and devices in the electrical network model. These calculations are performed on an electrical island basis by tracing from each energized source and collecting all the energized objects. SCADA measurements at the feeder head and at various points down the feeder are used to accurately distribute load to each load point. PFService sends the real-time power flow solution results, as well as information about voltage and flow violations, to various Oracle Utilities Network Management System windows for you to view.

CORBA Gateway Service

The CORBA Gateway service provides part of the interface between the Java-based applications such as Web Trouble, Storm Management, Web Call Entry, etc. and the other C++-based Oracle Utilities Network Management System services. The CORBA gateway allows the Java Application Servers to get published updates from services like JMService, DDService or MTService and also provides the mechanism to query these services directly on-demand. The Java Application Servers (*i.e.*, WebLogic) must then take these updates and make them available for the Java (end-user) client applications.

The CORBA Gateway service uses Isis to communicate with the other Oracle Utilities Network Management System services. The CORBA Gateway service requires that the TAO (TheACE ORB) CORBA Naming Service be running. Normally TAO is configured to run (be default) on startup. See the $CES_HOME/templates/tao.template script for an annotated example of a tao startup/shutdown script that could be configured to run at system startup.

> **Note:** We now recommend that you run two copies of the CORBA gateway for each Oracle Utilities Network Management System environment.

1. The first instance is a dedicated publisher instance that takes messages published via the Oracle Utilities Network Management System services and publishes them to the Java Application Server (WebLogic).
2. The second instance is dedicated to handling Java client application requests to Oracle Utilities Network Management System services.

   Examples of how to setup these corbagateway instances can be found in the $CES_HOME/templates/system.dat.template file.

The WebLogic Java Application Server must be configured to correctly connect to the appropriate corbagateway(s). See the Oracle Utilities Network Management System installation guide for instructions on configuring the Java Application server.

Service Alert Service

Service Alert processes updates from other services such as job/event update information, device operations, as well as receiving notifications from database triggers. These “updates” serve as the triggers for Service Alert to determine when the criteria for sending out a notification have been met. Once triggered, Service Alert gathers relevant data and sends out the desired notifications.

Service Alert Email Administration

How Service Alert Email and Paging Notification Work

When initiating a notification, Service Alert sends email and paging requests to the CORBA gateway. It is the email toolkit code within the CORBA gateway that interfaces with a mail system. The email toolkit uses SMTP to send these message requests. * Therefore, to properly receive Service Alert notifications, an SMTP server needs to be configured and running on the network. All that is left to do is to describe to the email toolkit the configuration settings that it needs in order to communicate with the SMTP server.

Note: Pager notifications are also sent by SMTP, since most major paging providers allow messages to be sent to a pager via an email aliasing system.

Entering Email/Pager Configuration Settings

The following Unix environment variables need to be set up properly in order to configure the email/pager notifications.

Variable	Description
CES_SMTP_SERVER	This is the fully qualified network hostname of the mail server.
CES_DOMAIN_SUFFIX	Domain Suffix – This value should be a valid domain such as “oracle.com”. This value is used in constructing the domain portion of the “From” field for all outbound messages. This field is also used during SMTP communication between the CORBA gateway and the mail server. It is important to set this to a valid domain, as some SMTP servers will verify that the domain exists and is real. If the server does not believe that the domain is legitimate, the email message may be discarded

The **Email Username** setting is a command line parameter on the CORBA gateway. The username is the string that appears after the “-username” command line option. This will appear in all email and pager notifications “From” field. It is probably a good idea to set up an email alias for this username, in case notification recipients attempt to reply to a notification. Note that the “@domain.com” portion of the username should be omitted as this comes from the “CES_DOMAIN_SUFFIX” Environment variable.

Service Alert Printing Administration

After installing the Oracle Utilities Network Management System Web Gateway, the following configuration steps need to be performed to allow printing from Service Alert.

Adding Printers for Service Alert

A Unix System Administrator will need to add the printers/queues to the Unix server where the Service Alert application is running.

Using the Update Printers Utility

The list of available printers will be kept in the MYC_PRINTERS table in the Oracle Utilities Network Management System database. Running the “UpdatePrinters” utility will populate this table. This utility is installed with the base Oracle Utilities Network Management System. UpdatePrinters utility should be executed in the environment where CORBA Gateway service runs. UpdatePrinters will look at the current environment to obtain a list of available printers. If the list of printers in the current environment does not match the contents of the MYC_PRINTERS table, the user will be asked if he would like to synchronize the table with the current environment. Also, for any contact that does not have a known printer, the tool will give the user the opportunity to change the printer location for that contact.

Services Configuration File

The Services Configuration Data File (system.dat) configures services for operation. It determines how services are defined, which default flags to use, on which computers, and how long the waitfor timer runs. The system.dat file is located in the $NMS_HOME/etc directory.

There are a number of sections in the system.dat file. The most critical sections include:

- scripts
- server
- services
- applications
- program
- instances

Scripts

The following table defines the scripts that SMService uses to perform various tasks.

Script	Description

LaunchScript	Used to launch a service. The most widely used mechanism for starting all the services is:
	sms_start.ces
	The default script to start a single service is sms_start_service.ces. Its syntax is:
	`sms_start_service.ces <host> <service> <process> <options>`
	host – Name of the machine on which to run the service
	service – Name of the service
	process – Name of the executable that launches the service
	options – Command line options that are passed to the process at initialization
	For example, to start DBService, type:
	`sms_start_service.ces train1 DBService DBService -nodaemon`
	Define the launch script in system.dat as follows:
	`LaunchScript <script name>`
	If no script is specified, then sms_start_service.ces is assumed.

Script	Description

Notify Script	Announces an event. This script eliminates the need for an Isis tool as an announcer. It can be used to generate e-mails and logs, or to interface to paging systems. When developing this script, keep in mind that it does not connect to Isis. The syntax is:
	`<script name> <time> <host> <process> <event type> <system state> <old system state> <message>`
	time – Date/time stamp.
	host – Name of the machine on which the processes are running.
	process – Name of the process.
	event type – The process state. Valid values are:
	STARTING – The process has started.
	INITIALIZING – The process has registered and is initializing.
	RUNNING – The process reports as initialized.
	FAILED – The process has failed.
	FAILED_INTERFACE – The process reports a failed interface.
	STOPPED – The process intentionally stops.
	INFO – The process generates a progress report.
	system state – State of the system. Valid values are:
	INITIALIZING – SMService is launching processes from system.dat.
	NORMAL – All processes are running or are intentionally stopped.
	WARNING – A non-critical process has failed. This state also refers to failed critical processes that have another instance running.
	CRITICAL – A critical process has failed and there are no other instances running.
	old system state – State of the system before the event generating the announcement occurred.
	message – Message supplied by SMService or the process that caused the event.
	Define the notify script in system.dat as follows:
	`NotifyScript <script name>`
	There is no default value, so if a script is not defined here, then only Isis announcements are generated.

CoreScript	SMService looks to this script for instructions when a core file is detected. This script determines what should be done with the file, such as announce the existence of the file, delete it, archive it, or e-mail the administrator. It does not connect to Isis. Its syntax is:
	`<script name> <process> <corefile>`
	process - Name of the process that has produced the core file
	corefile - Path to the core file
	Define the core script in system.dat as follows:
	`CoreScript <script name>`
	If a script is not defined, the core file remains and will be detected by SMService during the next cycle.

Server

This section of the system.dat file defines all machines that run services. Each server must be assigned a separate server ID number from 1 to 10. The format is:

```
service <hostname> <server id>
```

For example, for services running between machines london and paris:

```
server london 1
```

```
server paris 2
```

The value for hostname can be specified literally as <local>. If this is the case, then SMService will automatically substitute the name of the current node as the machine name. For example:

```
server <local> 1
```

While it is possible to configure services to run on different nodes and to have redundant versions of non-database services running on multiple nodes this is generally only done for very specific circumstances. In general it is suggested that you use the <local> syntax and run everything on one server.

Service

These entries in the system.dat file are definitions of services and process groups, such as interfaces, that are launched and monitored by SMService. Below is a sample service section:

# NAME	REQUIRED	START	DELAY	RESTARTS	RESET	MODE
service SMService	Y	60	0	10	86400	
service DBService	Y	90	0	10	86400	
service ODService	Y	180	0	10	86400	
service DDService	Y	180	0	10	86400	
service MTService	Y	180	0	10	86400	
service MBService	Y	180	0	10	86400	
service JMService	Y	280	0	10	86400	
service SwService	Y	280	0	10	86400	
service PFService	Y	4000	0	10	86400	
service corbagateway	Y	120	0	10	86400	
service service_alert	Y	120	0	10	86400	

The following table describes the SMService Service fields.

Field	Description
NAME	The name of the executable for the particular service.
REQUIRED	Indicates whether the instance of the service is required for the system to be functional. Valid values are 'Y', 'Yes', 'N', or 'No'. If there are no instances of a required service, the system locks until an instance is started.
START	The time taken for a service to start.
DELAY	Sets the number of seconds to wait before restarting a failed service. It only applies to processes that failed after they were running. Processes that fail before initialization are restarted based on the period parameter. A negative number indicates that the process is not restarted.
RESTARTS	The number of times to attempt restarting a process. A process is no longer automatically restarted after this value is exhausted until the process is reset (see below).
RESET	The timeout period that controls the rate at which processes are reset. When a process is reset, the restart counters re-initialize. A negative value deactivates this feature.
MODE	An optional argument that specifies the high availability mode of the service. If a mode is specified, the service starts with -<mode> and -number <n>, where <n> is the id defined for the node in the server line. Valid modes are exclusive, redundant, parallel or not specified. Exclusive runs with only one server. Redundant specifies running two servers, each with a database that mirrors the other. Parallel involves using Oracle Parallel Server to run two servers with a shared database.

Program

The program section of the system.dat file defines the executable program and command line options for each service. This section is optional, but can be used for the following:

- Specifying an alternative executable for a particular service. For example, setting TCDBService as an instance of DBService.
- Specifying command line options across all instances of a service. This simplifies the instance definition so that the command line options do not have to be duplicated for each definition.

Below is a sample applications section:

# NAME	EXE	ARGS
program DBService	DBService	-nodaemon
program ODService	ODService	-nodaemon -aggregates

# NAME	EXE	ARGS
program DDService	DDService	-nodaemon -zones -subscribezone -allowReset -alarms ALL
program MTService	MTService	-nodaemon
program MBService	MBService	-nodaemon
program JMService	JMService	-nodaemon -dbs
program SwService	SwService	-nodaemon
program PFService	PFService	-nodaemon
program corbagateway	Corbagateway	-nodaemon -ORBInitRef NameService=iioploc://<hostname>:1750/NameService -ORBLogFile /users/<username>/dialog_log/orb.log -ORBDebugLevel 3 -implname InterSys_<hostname>_<username> -iorfile /users/<username>/etc/<username>_vns.ior -publisher -xmldir /users/<username>/dist/wwwroot/xml
program service_alert	Mycentricity	-nodaemon -xmldir /users/<username>/dist/wwwroot/xml

The following table describes the SMService Program fields.

Field	Description
NAME	Specifies the name of the service that the executable belongs to. Valid services for this value are defined in the service section.
EXE	Specifies the name of the executable that runs the service.
ARGS	Defines the command line options that are used in all instances of the service.

Instance

The instance section of the system.dat file defines how the services are started. The format of each line is:

```
instance <node> <service> <database/args>
```

The following example starts nine services on the local node.

#	NODE	SERVICE	DATABASE/ARGS
instance	<local>	SMService	
instance	<local>	DBService	

#	NODE	SERVICE	DATABASE/ARGS
instance	<local>	ODService	
instance	<local>	DDService	
instance	<local>	MTService	
instance	<local>	JMService	
instance	<local>	SwService	
instance	<local>	corbagateway	
instance	<local>	service_alert	

The following table describes the SMService Instance fields.

Field	Description
NODE	Defines the node. Valid nodes for this value are defined in the server section. The value for NODE can be specified literally as <local>. If this is the case, then SMService will automatically substitute the name of the current node as the instance for which the service is to be started. By using "<local>" in place of a specific machine name, you can simplify your effort when replicating a system; you will not need to make changes to the system.dat at all.
SERVICE	The service being defined.
DATABASE/ARGS	Command line arguments that are applied when the service starts at this node. If the program section specifies command line options for a particular service, it applies to all nodes, so the arguments do not need specification here.

Model Build System Data File

The Model Build System Data File (system.dat.model_build) configures services for Model Build/ Configuration operations. It is formatted the same as system.dat.

The system.dat.model_build starts only SMService, DBService, ODService, and MBService. These services are generally executed from configuration scripts, such as ces_setup.ces, which require that some services be running to access the database and object classes.

Starting and Stopping Services

In order to start services, the following configuration files must be updated for the specific site configuration:

```
~/etc/system.dat.model_build
```

```
~/etc/system.dat
```

```
~/etc/system.dat.init
```

Starting Services

To start services, complete these steps:

1. Login to the server machine as the Oracle Utilities Network Management System Admin user.
2. Type:

```
sms_start.ces
```

SMService starts. It reads and caches the system.dat file by default and starts the remaining services based on the data it just cached.

> **Note:** Using the -f <filename> option with sms_start.ces will override the default behavior and SMService will cache the specified file instead (e.g., ~/etc/system.dat.init, or ~/etc/system.dat.model_build.etc.).

Stopping Services

To stop services, type:

```
sms_stop.ces -s
```

When stopping services, you may have other tools running. The services are the core dependencies of all applications, so when services are stopped, all tools should be stopped and then restarted after the services have been re-launched. The best method to stop everything short of stopping Isis is to stop the process by groups.

1. To stop both clients and services:

```
sms_stop.ces -a
```

> **Note:** Occasionally, there are tools or Isis processes that may continue to exist as defunct and/or hung processes after the above commands do (or do not) run to completion. Check the process list on the Unix machines for these processes and kill them prior to restarting. Otherwise, otherwise the system may not restart properly.

Chapter 8

Building the System Data Model

The Model Build process creates the operations data model that mirrors the utility company's Geographic Information System.

This chapter defines the configuration of the model builder and provides an overview of validating and testing tools. It includes the following topics:

- **Model Builder Overview**
- **Data Directories**
- **Model Configuration**
- **Customer Model - Logical Data Model**
- **Customer Model Views**
- **Model Build Process**
- **Model Manipulation Applications and Scripts**
- **Schematics**
- **In Construction Pending / Device Decommissioning (ICP)**
- **Auto Throw-Over Switch Configuration (ATO)**
- **Symbology**
- **Power Flow Data Requirements and Maintenance**
- **Catalog Tables**
- **Power Flow Service High Level Messages**
- **Spatially Enabling the Data Model for Advanced Spatial Analytics**
- **NMS CIM Import and Export Tools**
- **Model Build File Export to XML**

Model Builder Overview

The Model Builder Service (MBService) is used in building an Oracle Utilities Network Management System operations data model. The Oracle Utilities Network Management System operations data model is built using the customer's existing *as-built* data model, which is typically a Geographic Information System (GIS) or graphic files (*e.g.*, AutoCAD). Necessary enhancements are applied to the GIS data model to make the *real-time* data model.

When changes are made in the GIS, MBService then merges them into the Oracle Utilities Network Management System data model. In addition to maintaining the model database, MBService also generates map files that are loaded for visual inspection.

A single spatial grouping of data known as a partition passes through various stages during its incorporation into the Oracle Utilities Network Management System Operations Model:

- GIS Data Extraction – to extract the data from the GIS to Oracle's vendor neutral model preprocessor (MP) file format.
- Preprocessing – to produce model build (.mb) files used by the Model Builder.
- Model Build (MBService) – saves the information into the Oracle Utilities Network Management System Operations Model RDBMS and writes out a set of maps.

The Model Builder service (MBService) is responsible for managing structural changes to the core operations model. Structural changes are largely the creation, deletion, and modification of objects. Non-structural changes involve updating attribute information such as status values.

The core operations model describes a set of interconnected network components with graphical representations and managed statuses. The objects contained within the model are subdivided into partitions with interconnections of partitions managed through the use of boundary nodes.

This data model must initially be obtained from an external source (such as a GIS) to populate the core operations model. Once populated, the core operations model is the basis for support of system services and the construction of diagrams.

The real-time services typically load parts of the model during initialization. These services also update attributes of the model. The process of model edit involves the creation, update, and deletion of objects that require consequential updates within services.

Patches

Import Files are submitted to MBService for processing. Each set of transactions submitted to MBService is considered a model patch and is applied to the current model. Most often, a patch is generated when a single partition is submitted to MBService for building.

The lifetime of a patch includes the following:

- Initial creation of the patch either locally or externally.
- Addition of the patch to the core operations model, where the patch will either be applied and become part of the current operations model or will be deleted if there is a problem with the patch resulting from patch format errors or real-time issues in the operations model (i.e., deleting a device with a call or outage).

Data Directories

OPERATIONS_MODELS Directory

The data directory, which is owned by the Unix Oracle Utilities Network Management System services user, must be unique for each Oracle Utilities Network Management System implementation. This directory is also referred to by the $OPERATIONS_MODELS environment variable. A unique version of this directory must be present for each Oracle Utilities Network Management System data model. It contains the model map files that the Viewer picks up and loads for the operator. These maps must be consistent with the data model; variations will cause problems with the Viewer's display of maps.

For example, if the same $OPERATIONS_MODELS directory is shared by the PRODUCTION and TEST environments (which have separate data models), then the map files that PRODUCTION accesses can get corrupted by model builds done in the TEST system. Model builds can also cause discontinuity between the data maps that PRODUCTION sees and the database that PRODUCTION uses.

With Unix systems, the model builds performed on TEST may fail due to an inability to write the resulting maps to the PRODUCTION data directory in the first place, if permissions are established correctly. This is a common mistake in environment construction. When you replicate environments, confirm that important variables like $OPERATIONS_MODELS are defined correctly.

Every Unix Oracle Utilities Network Management System user must have a local copy of the $OPERATIONS_MODELS directory available for use by the Viewer. Once set up correctly, Oracle Utilities Network Management System maintains various remote copies of the map files by copying them from the database server as they are modified.

```
OPERATIONS_MODELS/
   SYMBOLS/
   VFONTS/
   errors/
      Patch<n>.log
   metafile ($MB_META_HOSTS)/
      *.mf * *.bmf
   patches/
      *.mb
      done/
         *.mb
   reports/
   *.mad
   *.mac
```

The following table describes the Model Builder directories and files.

Directory/ Files	Description
SYMBOLS	Contains the defined symbol sets for the presentation of all objects. (Convention only. May be moved elsewhere.)
VFONTS	Contains the set of supported vector fonts used in the presentation of scalable text. (Convention only. May be moved elsewhere.)
errors	Contains the output files of the model builder specifically related to errors and patch processing. The log files are named in Patch<patch_number>.log format.

Directory/ Files	Description
*.mac	Textual representations of the background maps. The background map files corresponding to the *.mad files. These files are used by the Viewer to present background graphic information (boundaries, roads, text, etc.).
*.mad	Textual representations of the electrical maps. The map files used by the Viewer when presenting graphic information correlated to the network information stored in the database. It is essential to keep the database and the maps synchronized to ensure proper presentation and map conductivity.
metafile directory (binary map files)	The metafile directory ($MB_META_HOSTS) contains a binary copy of the maps that have been built. Whenever possible, the metafile is used instead of the *.mad files because metafiles are smaller and much faster.
patches	Contains <mapname>.mb files and/or <mapname>.mbd directories. These files define the model build transactions that will be submitted to the model. Files are moved into the done subdirectory after they have been submitted.
reports	This directory contains difference reports, which list all changes being introduced into the model for each patch. These are only generated if MBService is running with the -report option.
error	Contains the log files of the model build patch commitment process. The log files are named in Patch<patch number>. log format.

Binary Map Files

The **`$MB_META_HOSTS`** directory, which is set in the **`.nmsrc`**, stores binary versions of all the `*.mad` and `*.mac` files. This directory is set in the .nmsrc as . The Model Build Service converts the `*.mad` and `*.mac` files into operating system (OS) specific binary files stored as `*.mf` and `*.bmf` files, respectively. The Viewer uses this directory to first search for an appropriate metafile version of the map to load before trying to load the text version. The binary version improves the performance of the Viewer loading the map.

> **Note:** The caveat here is that the metafiles are OS architecture specific. While the `*.mad` and `*.mac` files can be reproduced from a Solaris system to a Linux system (for example), the metafiles must be rebuilt against the Linux system architecture using the `BUILD_METAFILES.ces` script; this process is extremely fast.

For (Application Server) client environments, the Viewer only requires metafiles, so the `*.mad` and `*.mac` files only need to reside on the Services/Database server. The metafiles are all you need to distribute to client systems. Since the metafiles are significantly smaller in size than the `*.mad` and `*.mac` files, the distribution process has a lower impact to bandwidth.

Replicating an Oracle Utilities Network Management System

Essentially an entire Oracle Utilities Network Management System can be replicated across architectures without loss of data content. All you need to take into consideration are the metafiles, binary executables, and Isis executables, all of which are the only OS specific characteristics of an Oracle Utilities Network Management System. This keeps you from relying upon specific machine architecture.

Model Configuration

Model configuration requires a number of configuration parameters, scripts, and SQL files to be defined in order to fully set up an operational Oracle Utilities Network Management System. The following sections provide a checklist of the configuration settings that are required for a successful model build.

Define Environment Variables

Each user of the Oracle Utilities Network Management System must have an **`.nmsrc`** file in the $NMS_HOME directory. Edit the **`.nmsrc`** file toset the following required environment variables:

Environment Variable	Description
RDBMS_TYPE	Database type (ORACLE).
RDBMS_HOST	Database host machine. In the case of ORACLE_OCI, append ".world" to the machine name. Dependent upon the Oracle installation.
CMM_CELL	The name of the Isis communication "channel". All systems that have the same CMM cell value will communicate. Any value may be used, as long as all the interacting systems have the same value. Other non-interacting systems may not have this value.
CES_DATA_FILES	This environment variable is set to the directory where most configuration data files used by Oracle Utilities Network Management System software are installed. This includes *.dat, *.sym, *.cel files, among others.
NMS_ROOT	This environment variable is set to the directory where the top of the Oracle Utilities Network Management System installation occurs (*i.e.*, ~/nms).
CES_HOME	This environment variable is set to the product installation directory (*i.e.*, $NMS_HOME/product/1.10.0.0).
CES_LOG_DIR	This environment variable is set to the location of the service log files.
DATEMSK	This environment variable points to the path of the ces_datefmt file.
CES_SERVER	This environment variable contains the hostname of the Oracle Utilities Network Management System server.
CES_SMTP_SERVER	This environment variable points to an SMTP server where mail transactions can occur.

Environment Variable	Description
CES_SQL_FILES	This environment variable is set to the directory where most SQL files used by Oracle Utilities Network Management System software are installed.
CES_SITE	This environment variable contains a list of a set of configuration standards. Oracle defines the standard base configuration upon which customer configurations are built. The CES_SITE variable indicates which configurations to use. The setup process looks only for the files containing the values specified in this variable. The syntax is: `CES_SITE = "<project> product ces"` The first argument is the name of the customer or project. The last argument is the name of the default base configuration. There may be multiple configurations specified between the first and last arguments. When the system boots it processes the arguments from right to left, so it first loads the base configuration. Then it moves on to the previous argument and loads the associated configuration files if they exist. The process continues until each argument is processed.
OPERATIONS_MODELS	This variable specifies the directory into which the model will be built. That is, all maps and log files from the model build are located in this directory.
SYMBOLOGY_SET	This environment variable is set to the full path of the Oracle Utilities Network Management System symbol file <project>_SYMBOLS.sym.
CES_DATA_TABLESPACE	Contains the name of the primary Oracle tablespace. The installation and setup process uses it to better manage how database tables are set up.
CES_INDEX_TABLESPACE	Contains the name of the Oracle tablespace that is to be used for most indexes. The installation and setup process will attempt to put most indexes into this tablespace.

Configure Isis

Isis is the messaging backbone used by the Oracle Utilities Network Management System Operations Model, and it is required for every step of a model build. See **Isis Configuration** on page 4-1 for information about setup and configuration.

The CMM_CELL environment variable must be set uniformly over the network in order to communicate with programs on other machines.

To ensure Isis is running, type:

```
ps -ef | grep isis
```

Result: A pid (process id) is returned to confirm that Isis is running.

Verify Database Connection

Through the installation process the nmsrc file and the Oracle Wallet should be setup correctly so the ISQL.ces script can be run by an administrative user to connect to an interactive session of the database.

ISQL.ces can make a connection to the database. To verify that a connection is possible to the database, complete these steps:

1. From the <project> user name on the master server, type:

   ```
   ISQL.ces
   ```

   A database prompt ensures that the environment is set up correctly.

2. Type quit to exit the database connection.

DirectorySet Up

The model builder is primarily concerned with the tables within the selected database and the directory structure located under ${OPERATIONS_MODELS} as shown below.

Verifying Directory Set Up

A directory structure must be set up. To verify that it has been set up, type the following commands:

```
$ cd ${OPERATIONS MODELS}
```

```
$ ls
```

Result: A list of all directories will be displayed.

Setting up the Directory Structure

If the directory structure has not been set up, run the script ces_mb_setup.ces to configure it. It requires the OPERATIONS_MODELS environment variable to be set to the user's map data directory.

> **Note:** The ces_mb_setup.ces script is part of the model setup process, so the step listed here is redundant if this has already been completed.

The <project>_mb_setup.ces script creates and cleans the directory structure for customer specific model build setups.

The <project>_mb_preprocessor.ces script is called during the initial setup process to set up any additional directories or database tables that may be required by the model preprocessor. It is only required if special setup is needed.

Cleaning Up the Directory

If the data directory already exists from an obsolete data model, ces_mb_setup.ces -clean should be called to clean up all the residual files.

> **WARNING: If you run this script with the -clean option, you will delete the operational model.**

Define and Organize Classes

The operations model is designed around a class hierarchy. At the top of the hierarchy is the superclass, from which all other classes inherit attributes. The hierarchy may have multiple levels, each level having a parent/child relationship. The superclass is the only level that is always a parent and never a child.

Class Inheritance Definition

Classes and inheritance are defined and configured in the <project>_classes.dat and <project>_inheritance.dat files, respectively, located in the <project>/data directory. These files are loaded when the ces_setup.ces command is run to set up the data model.

These files can be individually loaded using the ODLoad command. The syntax to load classes.dat in Classes table via ODLoad is:

```
ODLoad -c <filename>
```

The inheritance relationships file, inheritance.dat, can be loaded into the INHERITANCE table via ODLoad. The syntax is:

```
ODLoad -i <filename>
```

In addition to these base class and inheritance files, special files may be included for dynamic condition classes (<project>_cond_classes.dat, <project>_cond_inheritance.dat) and classes required for the power flow application (<project>_pf_classes.dat, <project>_pf_inheritance.dat). These additional files would be supplemental to the base files and should not duplicate any entries.

Oracle includes some required classes within the ces_core_classes.dat file. These classes are required in order for the Oracle Utilities Network Management System to work properly. Their inheritance is defined in ces_core_inheritance.dat and is also required. None of the information in these files should be changed, removed, or duplicated.

Configure Attribute Table

The Oracle Utilities Network Management System attribute table is populated using <project>_attributes.sql. The user attribute table is populated using the <project>_schema_attributes.sql file.

Configure Control Zones

If you plan to use Oracle Utilities Network Management System control authority functionality, then all electrical devices should have an assigned Network Component Group (NCG). This is usually assigned in the source data or computed in the preprocessor.

Configure Symbology

Oracle Utilities Network Management System Viewer symbol information is stored in `<project>_SYMBOLS.sym` file. The <project>_ssm.sql file maps classes to the particular symbol. The symbology file build process has been standardized to build the run-time symbol file (`$NMS_HOME/<project>/data/SYMBOLS/<PROJECT>_SYMBOLS.sym`) from these symbol file sources in order of increasing preference:

1. `$CES_HOME/product/data/SYMBOLS/MASTER_SYMBOLS.sym`,
2. `$CES_HOME/i18n/data/SYMBOLS/MASTER_SYMBOLS.sym`,
3. `$NMS_HOME/`*`<project>`*`/data/SYMBOLS/`*`<project>`*`_DEVICE_SYMBOLS.sym`,
4. `$NMS_HOME/`*`<project>`*`/data/SYMBOLS/`*`<project>`*`_CONDITION_SYMBOLS.sym`.

The command, `nms-make-symbols`, will do the construction of the run-time symbology file and will make a backup of the resulting file if one existed prior to the execution of this script. Run nms-make-symbols before running nms-install-config to get your *<project>*_SYMBOLS.sym file up to date with the your latest configuration and NMS product release.

Service Configuration File

The sms_start.ces.ces script is used to start up Oracle Utilities Network Management System services. It normally reads the system.dat file to determine which services to start up and what arguments to give them. Before a model is built, this configuration must not be used, because it contains startup commands for the Dynamic Data Service (DDService), the Managed Topology Service (MTService), and the Job Management Service (JMService), none of which will execute until a model has been at least partially built. The model build process expects to find another configuration file, system.dat.model_build, in the same directory that has a more limited set of services. In addition, there is a system.dat.init file that starts up only the database service.

Verify Licensed Products File

The Automated Setup script (`ces_setup.ces`) and related .sql and .ces files will reference a *<project>*`_licensed_products.dat` file to properly configure the model to support the products you have licensed. This file is a text file and contains a list of the licensed Oracle Utilities Network Management System options. There is a template version of this file in `$CES_HOME/templates/licensed_products.dat.template`. The template should be copied to your `$NMS_CONFIG/sql` directory and renamed to a *<project>*`_licensed_products.dat` file. Then you should edit the file to uncomment the options you have licensed and are implementing. This edited template file should then be installed using the `nms-install-config` installation script prior to running the `ces_setup.ces` command.

The following table describes the product codes used in the template file:

Product	Description
amr	Generic AMR/AMI integration
bi	Oracle Business Intelligence for Utilities integration
crewcentricity	Web Workspace, Web Trouble, Web Call Entry, Web Callbacks
crewman	Trouble Management
crsi_gateway	Oracle Utilities CCB-NMS integration
datamart	Oracle Utilities Performance Datamart product
dynratings	Dynamic Line Ratings
flisr	Fault Location, Isolation, and Service Restoration
flm	Feeder Load Management
flm	Feeder Load Management
ivr_gateway	Generic Interactive Voice Recognition integration
mobile	MQ Mobile integration
mq_gateway	IBM MQSeries integration
mycentricity	Service Alert
network_analysis	Power Flow
oms_mwm	Oracle Utilities NMS-MWM integration
opf	Volt/VAr Optimization

Product	Description
opws	Operator's Workspace
powerflow	Power Flow, Volt/VAr Optimization, FLM, Suggested Switching
ss	Suggested Switching
stormman	Storm Management
switchman	Switching Management (not available in NMS 1.11.x; mutually exclusive with WebSwitching)
troubleman	Trouble Management
webgateway	web-based applications
WebSwitching	Web Switching Management

Run Automated Setup

Oracle has an automated process that sets up the database schema and directory structure. Any scripts, SQL files, or data files that are properly set up, named and installed will automatically get picked up and used by this process. The automated setup process will use various SQL files mentioned in this section to build the initial data model.

ces_setup.ces

The **ces_setup.ces** script must be run on the model build host machine, which is the machine on which MBService is running. This process loads scripts, SQL, and data files that are properly configured and installed. The script makes liberal use of ISQL.ces, which submits all SQL files to DBService to be run. The syntax is:

```
ces_setup.ces [[-clean [-noVerify] [-reset]] | [-offline]] [-showme]
[-o <logFile>] [-noInherit] [-debug] [-noMigrations] [-cust]
[-migrationOnly] [-cust_schema] [-help]
```

The following table describes the **`ces_setup.ces`** command line options.

Option Variable	Description
`-clean`	Destroys the current model in order to build a new model. A prompt requires the user to verify this option. After this, a rebuilt model will still retain and use the same internal device identifiers (handles). This is useful for continuity of reporting before and after a **`clean`** model build.
`-noVerify`	Bypasses the interactive verification prompt that opens for the **`-clean`** option.
`-reset`	Resets the generation of internal device identifiers (handles). If **`-reset`** is used with **`-clean`**, a model built afterward will not be relatable to the previous model, even though they may look the same.
`-offline`	Preserves the data model, but erases the real-time and historic information concerning the model, such as tags, permits or notes. Configuration changes made directly to the database may be lost. For example, a list of login users maintained with the *SqlX* tool would be replaced with the login users defined in the `CES_USER` configuration table located in *`<project>`*`_ceslogin.sql`.

Option Variable	Description
-showme	Prints the complete list in sequential order of scripts, SQL, and data files that are loaded or executed during the model build. Child scripts are indented in the list to easily identify parents. This option must be included in the database table or directory creation scripts in order to work properly.
-o <logFile>	If the **-o** parameter is specified, output will go to the log with the specified *logFile* name, except if **-o -** is used, in which case output will go to stdout.
-noInherit	Skips base configuration and loads only the customer's configuration. This environment variable contains a list of a set of configuration standards. Oracle defines the standard base configuration upon which customer configurations are built. Use this option with caution, as it deviates from the supported process.
-debug	Turns on debug; does nothing.
-noMigrations	Skips the automatic PR migration process. Use this option with caution, as it deviates from the supported process.
-cust	Updates the customers view after the setup is completed.
-migrationOnly	Only run model migrations - not configuration.
-cust_schema	Create the customer schema tables.
-help	Print this help.

ces_setup.ces Log File

ces_setup.ces automatically sends its output to a log file in **$CES_LOG_DIR**. The standard naming convention is:

setup.*<date>*.*<time>*.log

The log file named is amended when any combination of the -clean, -offline, or -showme parameters are used:

- setup_clean.*<date>*.*<time>*.log
- setup_offline.*<date>*.*<time>*.log
- setup_showme.*<date>*.*<time>*.log
- setup_clean_showme.*<date>*.*<time>*.log
- setup_offline_showme.*<date>*.*<time>*.log

When output is sent to a log file, a single line will be sent to the console indicating the name of the log file. The first line of the log file shows the arguments that were passed to **ces_setup.ces**.

CES_SITE Variable

The `CES_SITE` variable indicates configurations to use. The setup process looks only for the files containing the values specified in this variable. The syntax is:

```
CES_SITE="<project> product ces"
```

The first argument is the name of the customer or project. The last argument is the name of the default base configuration. There may be multiple configurations specified between the first and last arguments. When the system boots it processes the arguments from right to left, so it first loads the base configuration. Then it moves on to the previous argument and loads the associated configuration files if they exist. The process continues until each argument is processed.

The `noInherit` option makes sure that only the left-most configuration is loaded. Usually the left-most configuration is the customer's project-specific configuration based on Oracle's standard product configuration.

The setup process runs a large set of shell and SQL scripts that set up all aspects of the **NMS** model. The right-most value of the `CES_SITE` environment variable identifies a "base," or predefined configuration. By default, the setup process sets up the model in the predefined configuration. However, the setup script contains numerous "hooks" that when encountered, install project-specific configuration that overrides the base configuration.

For example, if project XYZ defines a base model stdbase, then the `CES_SITE` environment variable is set to "*`xyz`*` stdbase`." The stdbase configuration is used by default, with project-specific files overriding stdbase files when encountered. The stdbase configuration may contain a script `stdbase_mb_preprocessor.ces` that sets up the data model for the stdbase version of the preprocessor. Project XYZ uses a different preprocessor with a different setup. The **NMS** setup process has a hook for a *`<project>`*`_mb_preprocessor.ces` file, so any file of this form with the project prefix as specified by `CES_SITE` (in this case, *`xyz`*`_mb_preprocessor.ces`) is called in place of the stdbase version. The exact details are dependent upon the nature of the "hook" involved. Some hooks are set up to call both the project script and the base script, while others will only call one or the other.

Linking In Customers

In order for Oracle Utilities Network Management System Trouble Management to run, user information must be linked into the model. This information is assumed to be in the database, whether explicitly loaded or whether linked in as a synonym. Oracle requires that the table that contains the end user information be joined to the SUPPLY_NODES table in the CES_CUSTOMERS table

Population of the CES_CUSTOMERS Table

The CES_CUSTOMERS table is populated with details about customers, their meters, and their locations. It includes information from the following tables:

- CU_CUSTOMERS
- CU_SERVICE_LOCATIONS
- CU_METERS
- CU_SERVICE_POINTS

To update the Oracle Utilities Network Management System customer model, project-specific customer import processes will drop and rebuild mirror versions of these tables, named:

- CU_CUSTOMERS_CIS
- CU_SERVICE_LOCATIONS_CIS
- CU_METERS_CIS
- CU_SERVICE_POINTS_CIS

They will then run `product_update_customers.ces`, which will perform change detection between the CU_*_CIS tables and their Oracle Utilities Network Management System counterparts, perform incremental updates to them, and re-create the CES_CUSTOMERS table.

> **Note**: If you do not want to update the CU_* tables or you do not have an updated set of CU_*_CIS tables, you should add the "`-no_pre_process`" option to the call to `product_update_customers.ces` and the CU_* tables will remain unchanged.

From the CES_CUSTOMERS table, a smaller table must be extracted that is called CUSTOMER_SUM.

Population of the CUSTOMER_SUM Table

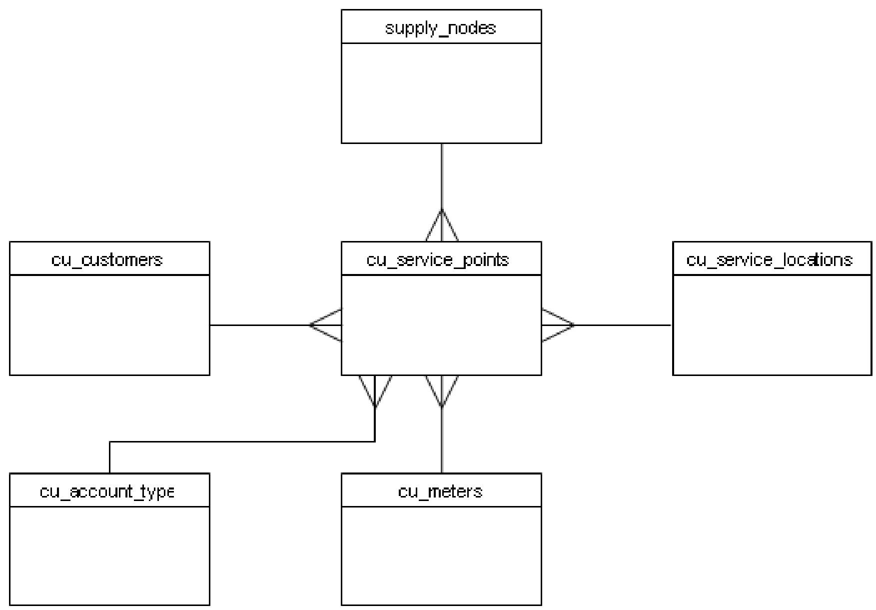
The CUSTOMER_SUM table is a smaller extraction of the CES_CUSTOMERS information in which the customer information is summarized. JMService uses this for faster calculations. Depending on the definition of CUSTOMER_SUM table, a fresh extraction may be required after each model build.

Customer Model - Logical Data Model

This section provides an overview of the logical view of the Oracle Utilities Network Management System customer model. Where the MultiSpeak data model uses Customer, Service Location and Meter entities, the Oracle Utilities Network Management System model adds the notion of a Service Point to increase flexibility, and provide for improved performance of the physical implementation. Additionally, the Oracle Utilities Network Management System model extends beyond the basic MultiSpeak model in the following ways:

- Supports more than one meter per service location.
- MultiSpeak attributes not required for NMS purposes are not required, such as billing information (acRecvBal, acRecvCur, …) and meterology information (kwh, multiplier, …)
- Provides model extensions to support important attributes not currently defined by MultiSpeak but necessary for NMS purposes.
- Supports customer-defined attributes for read-only purposes with no requirement for use in analysis.

This model, when joined with the Supply Node information in the Oracle Utilities Network Management System database (supply_nodes), results in the following E-R diagram:

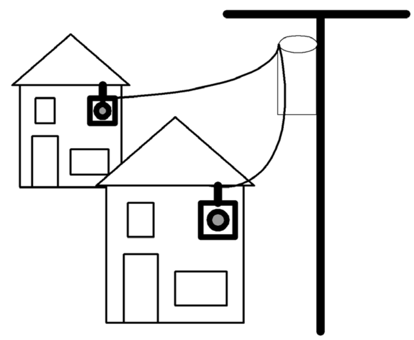


Residential Model

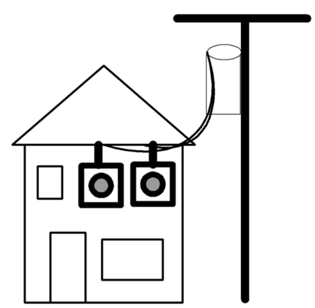
In this extended model, it is recognized that the occurrence of multiple meters is reasonably common, where each meter may have different rate codes associated.

Although the occurrence of multiple transformers is much less frequent than multiple meters, there are also several possible configurations of meters and transformers, with different electrical arrangements. Often, multiple transformers will occur on (geographically) large sites (e.g., factory, airport, shopping mall, etc.), where it is appropriate and helpful (from the perspective of outage analysis) to have multiple service locations defined for the site which aid in readily locating the appropriate transformer.

The following pictures depict some simple examples of the usage of this customer model. The first example shows two service locations, each with a meter connected to a distribution transformer.



The second example is an account with a single location with two meters, which is described through the definition of a customer account, a service location and two meters. The service location is associated with a distribution transformer.



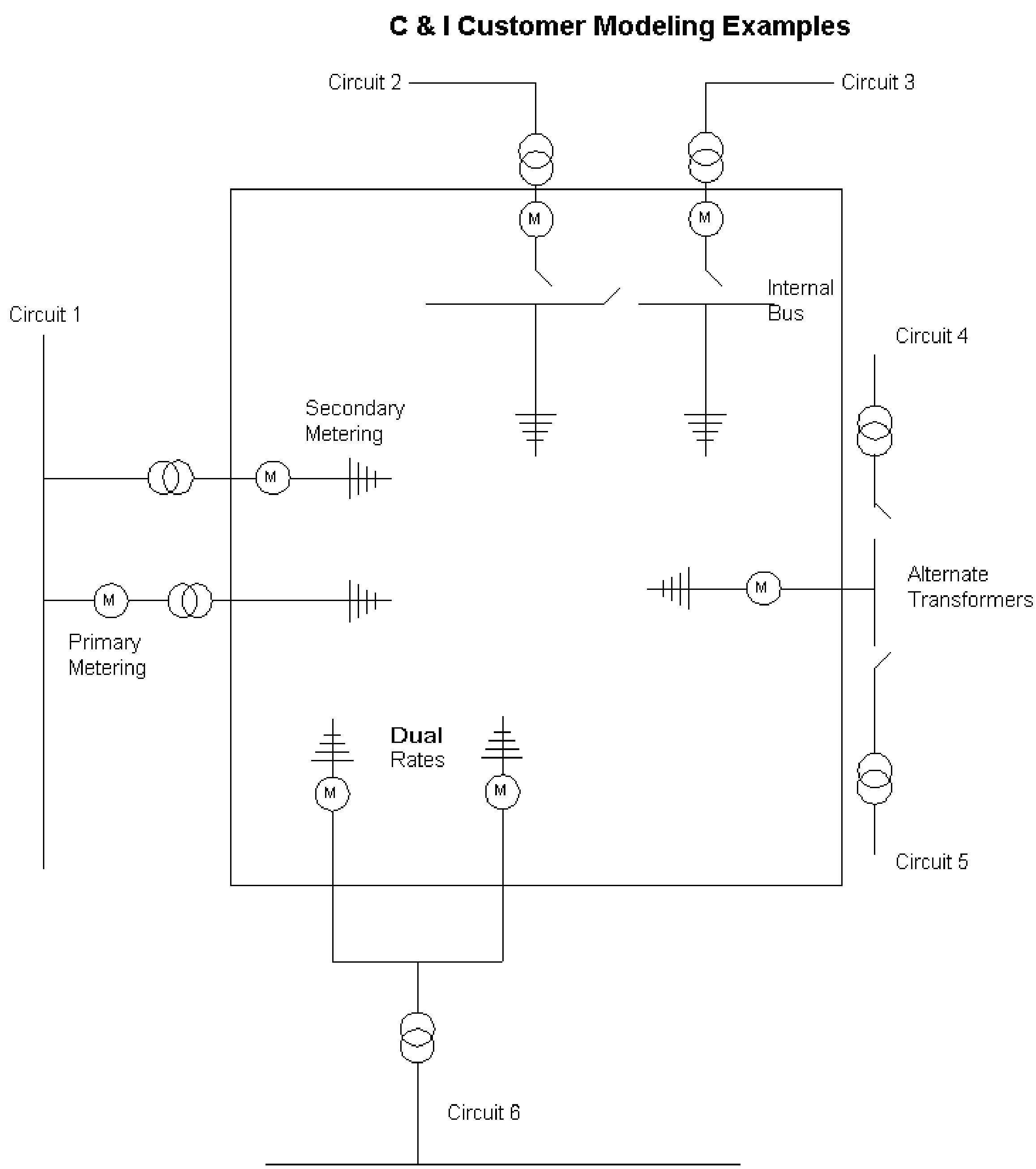
A third example would be a combination of the two previous examples, where a single customer account was responsible for the billing related to all of the above service locations. A more sophisticated example of residential metering is provided in the appendix.

Commercial and Industrial (C & I) Model

Many Commercial and Industrial situations are more complicated than residential metering. In these cases, a variety of configurations of meters, transformers and circuits must be addressed. The variations include:

- Primary metering, where the meter is placed on the high side of the transformer
- Internal buses, where two transformers can be used with two meters, feeding an internal bus
- Alternate transformers, where a meter can be switched to one of two transformers, each on a different circuit
- A single transformer feeding two meters, where different rates apply to each meter

The following diagram illustrates these examples.



Customer Model Database Schemas

The following section provides schema descriptions for the data and tables that are relevant to the Customer Model. It should be noted that the naming convention used internally is slightly different than the convention used in MultiSpeak or CIM exchange formats, due to the case-insensitive nature of Oracle RDBMS.

Customer Model Database Tables

The purpose of this section is to provide descriptions of the data and tables that support the implementation of the Oracle Utilities Network Management System customer model. These descriptions address only the data elements that are relevant to the customer model. The actual database tables may contain additional fields, but the other fields are not relevant to the customer model and are not described here.

Req Key Values	Meaning	Comment
N	Not required	Not needed for standard ces_customers table.
C	Configured in standard ces_customers table.	Not all columns referenced in the ces_customers table are required for a given implementation - inclusion of some columns can be project-specific.
Y	Required	Used in standard ces_customers table – still may not be 100% required. Actual requirements are generally project specific.

Customers Table

The **cu_customers** table is used to manage customer accounts. While the primary key is cust_id, this typically may have the same value as account_number.

Req	Column Name	Data Type	Description
Y,C	cust_id	NUMBER NOT NULL,	Primary key – may be generated.
N	cust_account_number	VARCHAR2(30) NOT NULL	Customer account number.
N	cust_billing_account	VARCHAR2(13) NULL	Customer billing account number.
Y,C	cust_name	VARCHAR2(90) NULL	Name of the customer; concatenation of last, first and middle names, or business name.
N	cust_last_name	VARCHAR2(30) NULL	Last name.
N	cust_first_name	VARCHAR2(30) NULL	First name. Typically, this is only populated for residential customers.
N	cust_middle_name	VARCHAR2(30) NULL	Middle name or initial.
Y,C	cust_home_ac	NUMBER(3) NULL	Phone area code for the home phone.
Y,C	cust_home_phone	NUMBER(7) NULL	Phone number for the home phone.
N	cust_day_ac	NUMBER(3) NULL	Phone area code for the work phone.
N	cust_day_phone	NUMBER(7) NULL	Phone number for the work phone.

Req	Column Name	Data Type	Description
N	cust_day_phone_ex	NUMBER(7) NULL	Typically, day phone numbers are related to customers' work phone numbers, which generally include extensions.
N	cust_bill_addr_1	VARCHAR2(50) NULL	Street address of the billing address. Note that billing address fields are usually populated only if different from the address held in the cu_service_point table.
N	cust_bill_addr_2	VARCHAR2(50) NULL	Second line, if necessary, of street address of the billing address.
N	cust_bill_addr_3	VARCHAR2(50) NULL	Third line, if necessary, of street address of the billing address.
N	cust_bill_addr_4	VARCHAR2(50) NULL	Fourth line, if necessary, of street address of the billing address.
N	cust_bill_city	VARCHAR2(30) NULL,	City of the billing address.
N	cust_bill_state	VARCHAR2(30) NULL	State of the billing address.
N	cust_bill_postcode_1	VARCHAR2(10) NULL	First 5 zip code numbers for US.
N	cust_bill_postcode_2	VARCHAR2(10) NULL	Second 4 zip code numbers for US.
C	cust_name_initials	VARCHAR2(3) NULL	The customer initials. Possibly used for certain soundex type searching if a customer wants to enable it - not often. Not necessary.
N	cust_comment	VARCHAR2(255)NULL	General field provided to support additional information about the customer, such as 30ft ladder, assult-case, crit-pmp-station, etc.
N	cust_user_def_1	VARCHAR2(255) NULL	These user-defined fields support the inclusion of other desired data not covered in the core fields. These fields can be extracted for project specific reporting.
N	cust_user_def_2	VARCHAR2(255) NULL	
N	cust_user_def_3	VARCHAR2(255) NULL	
N	cust_user_def_4	VARCHAR2(255) NULL	
N	last_update_time	DATE	Time of last update for record. Generally, set internally when a record is updated - not via external CIS.

Service Locations Table

The purpose of the cu_service_locations table is to manage locations (premises) at which a customer is served. A customer account may have multiple service locations.

Req	Column Name	Data Type	Description
Y,C	serv_loc_id	NUMBER NOT NULL	Primary key – may be generated.
N	serv_type	VARCHAR2(2) NULL	The type of service at this location. (electrical or gas). Only necessary for utilities that support multiple service types.
N	serv_status	VARCHAR2(50) NULL	Electrical service status of the service location. For example: INA – Inactive ACT – Active PDI – Pending Disconnect Can be used to coordinate business processes around how to handle customer disconnects (for example, update the day before). Each project needs to discuss these.
Y,C	serv_account_number	VARCHAR2(30) NOT NULL	The service account number which will be used for call entry purposes, and the account number used in createIncident XML.
Y,C	serv_revenue_class	VARCHAR2(30) NULL	Revenue class for the service location.
N	serv_load_mgmt	NUMBER NULL	Binary - whether or not there is load mgmt at this Service Location
Y,C	serv_concat_address	VARCHAR2(200) NULL	Concatenated address of the service address 1, 2, 3, and 4.
N	serv_special_needs	VARCHAR2(1) NULL	Identifies any special needs of the customer.
N	serv_priority	VARCHAR2(32) NULL	Mapped to ces_customers.priority. This defines the meaningful customer type value the utility uses internally. This value will be displayed on troubleInfo as well.
N	serv_addr_1	VARCHAR2(50) NULL	First line of street address of the service address.
N	serv_addr_2	VARCHAR2(50) NULL	Second line, if necessary, of street address of the service address.
N	serv_addr_3	VARCHAR2(50) NULL	Third line, if necessary, of street address of the service address.
N	serv_addr_4	VARCHAR2 (50) NULL	Third line, if necessary, of street address of the service address.
N	serv_city	VARCHAR2(25) NULL	City of the service location.
N	serv_state	VARCHAR2(25) NULL	State of the service location.
Y,C	serv_city_state	VARCHAR2(50) NULL	This field contains the data that will appear in the ces_customers.CITY_STATE field.
Y,C	serv_postcode_1	VARCHAR2(10) NULL	First 5 Zipcode numbers for US.
N	serv_postcode_2	VARCHAR2(10) NULL	Second 4 Zipcode numbers for US.

Req	Column Name	Data Type	Description
N	serv_user_geog_1	VARCHAR2(25) NULL	User geo codes typically used for political areas, such as counties, tax districts, etc.
N	serv_user_geog_2	VARCHAR2(25) NULL	
Y,C	serv_town	VARCHAR2(3) NULL	The town or county for the customer.
Y,C	serv_str_block	VARCHAR2(20) NULL	Block number - used in searches.
N	serv_str_pfix	VARCHAR2(10) NULL	The 'R' in R 321 Rolling Rd (R rear, F front, A adjacent, etc.)
Y,C	serv_str_struc	VARCHAR2(20) NULL	Structure relates to apartments, units, piers, docks, warehouse, slip, etc.
N	serv_str_name	VARCHAR2(30) NULL	Name of the street (Main Street).
N	serv_str_cdl_dir	VARCHAR2(10) NULL	Cardinal direction (N, S, E, W).
N	serv_str_sfix	VARCHAR2(10) NULL	ST, PKY, PLC, DR, RD, AVE, etc.
Y,C	serv_lot	VARCHAR2(10) NULL	Lot number – used in searches.
Y,C	serv_apt	VARCHAR2(8) NULL	Apartment number.
N	serv_elec_addr	VARCHAR2(50) NULL	Elec address used in searches.
N	serv_sic	VARCHAR2(8) NULL	Standard Industrial Code.
N	serv_comment	VARCHAR2(255) NULL	General comment about the service location.
Y,C	serv_cumulative_priority	NUMBER NULL	Summation of priority codes for this location.
Y,C	serv_life_support	NUMBER NULL	Indicates if this is a life-support customer.
Y,C	serv_d_priority	NUMBER NULL	D customer defined flag, 0 or 1 – often medical customers.
Y,C	serv_c_priority	NUMBER NULL	C customer defined flag, 0 or 1 – often emergency customers.
Y,C	serv_k_priority	NUMBER NULL	K customer defined flag, 0 or 1 – often key/critical customers.
Y,C	serv_map_loc_x	NUMBER NULL	GPS lat/long or other mapping coordinates.
Y,C	serv_map_loc_y	NUMBER NULL	
N	serv_user_def_1	VARCHAR2(255) NULL	These user-defined fields support other desired data not covered in the core fields. These fields can be extracted for project-specific reporting purposes.
N	serv_user_def_2	VARCHAR2(255) NULL	
N	serv_user_def_3	VARCHAR2(255) NULL	
N	serv_user_def_4	VARCHAR2(255) NULL	
N	last_update_time	DATE	Time of last update for record.

Meters Table

The cu_meters table describes meters that might exist at a service location. The use of meters is optional (but increasingly common) within Oracle Utilities Network Management System. Meter information is required for a project which intends to utilize integration with an Automated Meter Reading Infrastructure.

The cu_service_points table tracks the relationship between a meter (cu_meters) and a customer account (cu_customers) and service location (cu_service_locations).

Req	Column Name	Data Type	Description
Y,C	meter_id	NUMBER NOT NULL	Primary key – may be generated.
Y,C	meter_no	VARCHAR2(20) NOT NULL	Meter number.
N	meter_serial_number	VARCHAR2(20) NULL	Serial number on the meter.
N	meter_type	VARCHAR2(20) NULL	Type of meter (gas, electric, water, etc.).
N	meter_manufacturer	VARCHAR2(20) NULL	Manufacturer of the meter.
N	meter_phases	VARCHAR2(1) NULL	Phase(s) connected to the meter (IE 1, 2, or 3).
N	meter_rate_code	VARCHAR2(65) NULL	Rate code for the meter.
N	meter_user_def_1	VARCHAR2(255) NULL	These user-defined fields support other desired data not covered in the core fields.These fields can be extracted for project-specific reporting purposes.
N	meter_user_def_2	VARCHAR2(255) NULL	
N	meter_user_def_3	VARCHAR2(255) NULL	
N	meter_user_def_4	VARCHAR2(255) NULL	
Y,C	meter_amr_enabled	VARCHAR2(1)NULL	'Y' or 'N' – REL_10_0.
N	last_update_time	DATE	Time of last update for record.

Account Type Table

The purpose of the cu_account_type table is to contain a configuration of the valid Account Types that can be specified for a Service Point record. The initial loading of customer data populates this table. There is often only one row in this table (for electrical service).

cu_account_type		
Column Name	**Data Type**	**Description**
acctyp_account_type	VARCHAR2(10) NOT NULL	Electric, Gas, Propane, Appliance Repair, etc.
acctyp_user_def_1	VARCHAR2(255) NULL	These user-defined fields support other desired data not covered in the core fields. These fields can be extracted for project-specific reporting purposes.
acctyp_user_def_2	VARCHAR2(255) NULL	
acctyp_user_def_3	VARCHAR2(255) NULL	
acctyp_user_def_4	VARCHAR2(255) NULL	

Service Points Table

The purpose of the cu_service_points table is to manage the linkages between the cu_customers, cu_service_locations, cu_meters, cu_account_type and supply_nodes tables.

Key indexes are placed on this table for performance. History can be tracked, by setting active_fl to 'N' to identify that a record is now historical. No timestamp is used to track when a service point went out of service and the cu_service_points table is not intended nor recommended as a long term repository for service point history.

Req	Column Name	Data Type	Description
Y,C	serv_point_id	VARCHAR2(64) NOT NULL	Primary key. If the CIS cannot provide a unique value, use a generated key (for example, by combining cust_id, serv_loc_id and meter_id columns). This is used for CIS-to-NMS integration. For Customer Care & Billing (CC&B) integration in Oracle Utilities Network Management System 1.10, this is the CC&B Service Point Id. (See below for related info on ces_customers).
Y,C	cust_id	NUMBER NOT NULL	Foreign key ref to the cu_customers table.
Y,C	serv_loc_id	NUMBER NOT NULL	Foreign key ref to the cu_service_locations table.
Y,C	meter_id	NUMBER NOT NULL	Foreign key ref to the cu_meters table.
Y,C	device_id	VARCHAR2(25) NOT NULL	Foreign key ref to the supply_nodes table. This field is critical and necessary, as it ties Oracle Utilities Network Management System to the CIS.
N	feeder_id	VARCHAR2(10) NULL	Foreign key ref to the supply nodes table. Note this field is non-critical and generally not necessary.
Y,C	active_fl	VARCHAR2(1) NOT NULL	Identifies currently active records. Generally, this is always ‘Y’ as there is little provision or need for inactive records in the system. Inactive records are generally removed from this table.
N	create_dttm	DATE NOT NULL,	Timestamp for the record's creation.
Y,C	account_type	VARCHAR2(10) NOT NULL	Foreign key to the cu_account_type table.
N	last_update_time	DATE	Time of last update.

Linkages to Other Tables

The customer model has linkages to other tables in the Oracle Utilities Network Management System model. The primary linkage between utility customers and the Oracle Utilities Network Management System electrical network model is the *device_id* column. The definitive table linkage is between supply_nodes.device_id and cu_service_points.device_id. From the perspective of the cu_service_points table, the device_id field is used to uniquely identify the electrical network model element (supply node) which supplies power to a service point (customer).

In general, an Oracle Utilities Network Management System *supply node* is any place on the model where a utility customer can be connected to receive electrical power. For customers that wish to model secondary network, this supply point can be associated with a single customer/meter. For customers that are only interested in modeling primary distribution circuits, the supply node is often associated with a secondary transformer.

The Oracle Utilities Network Management System electrical data model is implemented under the assumption that the source for the electrical network model data (generally a Geographic Information System) and the source for the utility customer data (generally a Customer Information System) understand and maintain this customer-to-supply-node relationship. The accuracy of this linkage is critical for reliable trouble call handling and outage reporting. Without this linkage, customer trouble calls enter the system as fuzzy calls and outage reports have diminished accuracy.

Customer Model Views

The purpose of this section is to describe the views that support existing Oracle Utilities Network Management System software, and provide compatibility for this customer model with existing installations.

CES Customers View

The ces_customers view (or table) is derived from the cu_customers, cu_service_locations, cu_meters, cu_service_points and supply_nodes tables. It provides a flat customer view that is utilized by various Oracle Utilities Network Management System services and applications such as JMService, Web Call Entry and others.

Displayed Column Name	Originating Table	Column in originating table
id	cu_service_points	serv_point_id
h_cls	supply_nodes	device_cls
h_idx	supply_nodes	device_idx
supply_idx	supply_nodes	h_idx
meter_number	cu_meters	meter_no
device_id	supply_nodes	device_id
account_type	cu_service_points	account_type
account_number (not null)	cu_service_locations	serv_account_number
account_name	cu_customers	cust_name
address_building	cu_service_locations	serv_str_struc
block	cu_service_locations	serv_str_block
address	cu_service_locations	serv_concat_address
city_state	cu_service_locations	serv_city_state
zip_code	cu_service_locations	serv_postcode_1
phone_area	cu_customers	cust_day_ac
phone_number	cu_customers	cust_day_phone
priority	cu_service_locations	serv_cumulative_priority
c_priority	cu_service_locations	serv_c_priority
k_priority	cu_service_locations	serv_k_priority
d_priority	cu_service_locations	serv_d_priority
life_support	cu_service_locations	serv_life_support
avg_revenue	cu_service_locations	serv_revenue_class
name_initials	cu_customers	cust_name_initials
town	cu_service_locations	serv_town
feeder_id	supply_nodes	feeder_id

Displayed Column Name	Originating Table	Column in originating table
lot	cu_service_locations	serv_lot
apt	cu_service_locations	serv_apt
cust_id (not null)	cu_customers	cust_id
meter_id (not null)	cu_meters	meter_id
serv_loc_id (not null)	cu_service_locations	serv_loc_id
amr_enabled	cu_meters	amr_enabled
x_coord	cu_service_locations	serv_map_loc_x
y_coord	cu_service_locations	serv_map_loc_y
user_geographic_log	cu_service_locations	serv_user_geog_1

Customer Sum View

Within Oracle Utilities Network Management System, the customer_sum view (or table) is used primarily by JMService to identify the number of customers, critical customers, etc. on each supply node. The customer_sum view/table is typically generated from the ces_customers table/view. It is simply a summation of the customer model and is designed to provide more efficient outage impact estimates.

Displayed Column Name	Originating Table	Column in originating table
supply_cls	supply_nodes	h_cls (=994)
supply_idx	supply_nodes	h_idx
device_id	supply_nodes	device_id
revenue	cu_service_locations	serv_revenue_class
customer_count	count(distinct cu_service_points)	cust_id
critical_c	sum(cu_service_locations)	serv_c_priority
critical_k	sum(cu_service_locations)	serv_k_priority
critical_d	sum(cu_service_locations)	serv_d_priority
critical_both	sum(cu_service_locations)	Combination of either critical c, critical k, critical d types. (serv_cumulative_priority)
x_coord	point_coordinates	x_coord
y_coord	point_coordinates	y_coord
ddo		Historical – likely should be removed at some point. Often set the same as customer_count to satisfy JMService.

zip_code	ces_customers	zip_code
city_state	ces_customers	city_state
user_geographic_loc	ces_customers	user_geographic_loc

Model Build Process

Model Build with a Preprocessor

In most cases, customers will place source data files into a designated directory and run the `ces_model_build.ces` script. This script takes no arguments and builds whatever maps are recognized by the `<project>_maps_to_build.ces` script. When the build completes, any completed maps will have import files automatically placed in a designated directory. In some cases, models may be built directly from import files.

Customer Model Build Scripts

The following table describes the model build scripts.

Script	Description

ces_model_build.ces	Builds the maps recognized by `<project>_maps_to_build.ces`. Upon completion, the `${OPERATIONS_MODELS}/patches/done` directory contains import files for the built maps.
	
	The script will check for a `<project>_model_build.ces` script, if it exists, it will be called instead of running the rest of the ces_model_build.ces script.
	
	Options
	`-noprebuild`
	`-nopostbuild`
	
	The parameters will cause the ces_model_build.ces script to skip prebuild script execution or postbuild script execution.

<project>`_model_build.ces`	If a `<project>_model_build.ces` script exists, it will be called by `ces_model_build.ces` and will be used as the project configured script to run the model build process. Most projects will not use this script.
*<project>*_build_map.ces	Required for any model build process that has a model preprocessor. Takes a map name and generates an import file for that map. The resulting import file is placed in the $OPERATIONS_MODELS/patches directory.
ces_build_maps.ces	This script takes multiple map prefixes as parameters. Any maps supplied will be built as a single model transaction. This is recommended when there is a model transaction involving multiple maps, especially if facilities are being transferred from one map to another. This does not run any _prebuild or _postbuild scripts.

Script	Description
<project>_maps_to_build.ces	Required for all model build processes. Identifies and prints a list (single line, space separated) of all maps that are queued up to be built. Model .mb files in the patches directory should be included in the list of maps to build including the .mb extension. All other maps to build should be reported without extensions.
<project>_postbuild.ces	Although not a required element of the model build, project-specific needs may call for an additional process after each model build. The additional process is carried out by the <project>_postbuild.ces script. It is run after the ces_model_build.ces script builds a complete set of maps. Common reasons for this process include recalculations of control zones, a fresh extraction of the CUSTOMER_SUM table, or a recache of DDService.
<project>_prebuild.ces	Although not a required element of the model build, project-specific needs may call for an additional process before each model build. The <project> _prebuild.ces script carries out the additional process. It is run before the ces_model_build.ces script builds a complete set of maps. This process is rarely needed.

Model Build with a Post-Processor

If a post-processor is needed for the model build, you should create and install the <project>_postbuild.ces script. If the post-processor requires patches to be applied to the model, it will build import files and put them in the patches directory. The ces_build_map.ces script can be called with the -noVerify option to build each patch without user interaction.

Constructing the Model

To ensure correct model construction, complete these steps:

1. To ensure Isis is running, type:

   ```
   ps -ef | grep isis
   ```

   Result: A pid (process ID) should be returned confirming that Isis is running.

2. If a model build preprocessor is being used, make sure that the expected scripts are created and installed. These are `<project>_build_map.ces` and `<project>_maps_to_build.ces`.

3. When new files are brought to the system, place them in the appropriate directory on the master server before initiating the model build.

   - Import files should go into the `${OPERATIONS_MODELS}/patches` directory.
   - Preprocessor input files will probably go into a project-specific directory. An example of a commonly used directory is `${OPERATIONS_MODELS}/mp`.

4. Log into the master server as the administrative user and initiate a model build by typing:

   ```
   ces_model_build.ces
   ```

   Or, if you want to produce a build log, enter the following:

   ```
   ces_model_build.ces | tee
     model_build.$(date "+%Y%m%d.%H%M%S").log 2>&1
   ```

Result: Each import file will be processed, updating the Operating Model and Graphic Presentation files.

5. Wait for the user prompt before continuing further model build operations. This process may take some time.
6. Review the error output information contained in the errors directory.

The Model Build Preprocessor

Oracle Utilities Network Management System obtains descriptions of the physical, electrical, and topological infrastructure from CAD, GIS and AM/FM systems through the model builder and associated preprocessors. The purpose of a preprocessor is to extract information from a source (GIS, CAD, AM/FM, etc.) and convert it to the neutral Oracle Utilities Network Management System import (.mb) format. From this format, it is processed by the model builder to determine and apply actual changes to the Oracle Utilities Network Management System operations model.

When the product is to be configured for a customer, there is a need to populate the corresponding Operations Model. Typically customers will have data stored within one or more forms: within a GIS, within a CAD product, in an RDBMS, or in flat files.

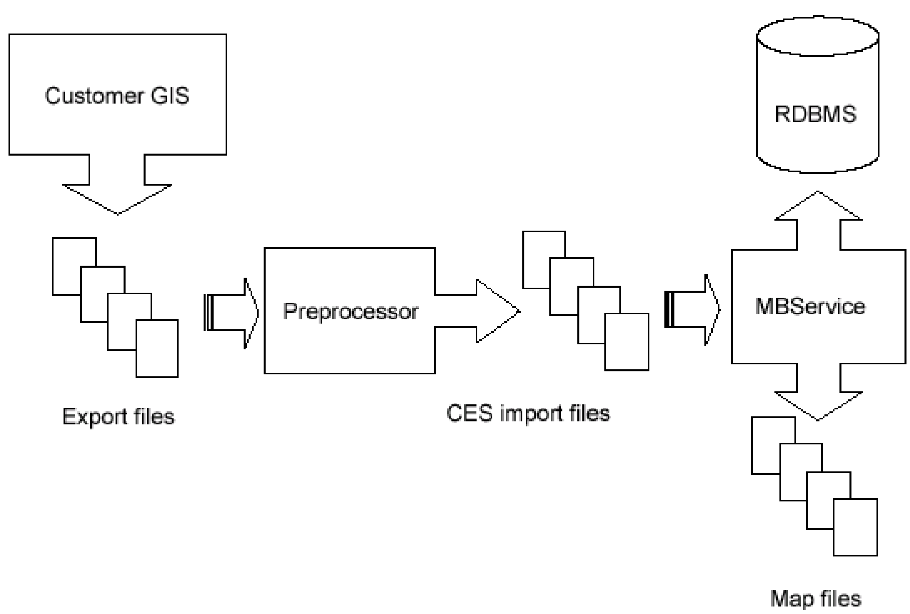
The information within these forms can either be directly extracted or preprocessed to a form which can be presented to the Model Build interface.

Model Build Basics

Model Build is a process of steps that will generate an operational topological representation of client's existing GIS. A single segment of data (partition) passes through four stages during its incorporation into the Operations Model:

- **Extraction**
- **Preprocessing**
- **Model Build** (MB Service)
- **Completed Operations Model**

The following figure provides an overview of the model build process:



Extraction

The graphical representations of objects that will be modeled, along with the associated attributes, are grouped and exported into external files in a format that the preprocessor is capable of reading. It is at this stage that the partitioning of the model into geographic grids or schematic diagrams is typically determined.

Preprocessing

The preprocessor reads the files generated by the extraction process and constructs an Import file which models the extracted portion. The preprocessor tends to be a major development task, taking weeks or months to complete.

Model Build

The Model Build (or MB Service) parses the Import file, verifies basic model consistency, applies the contained changes to the Operations Model Database, and commits the changes as part of the final model.

Completed Operations Model

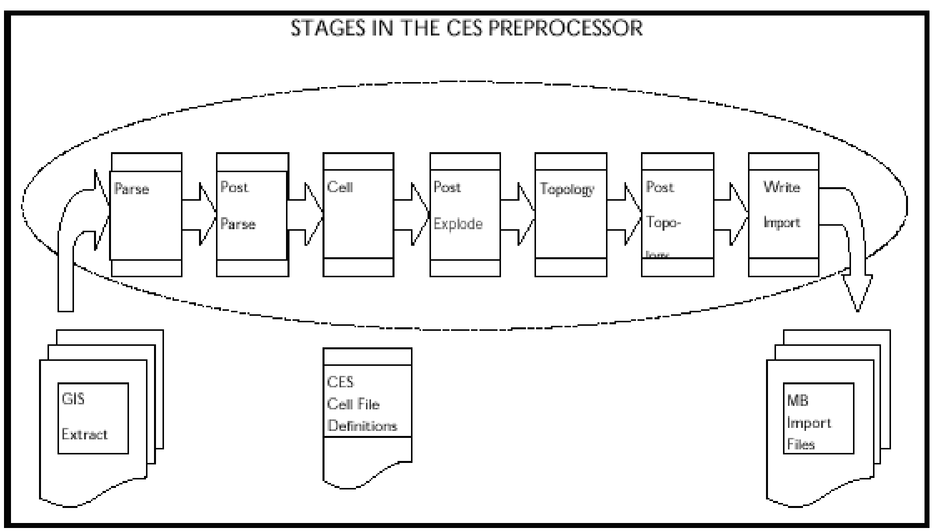
The completed model consists of new or updated partitions and new or revised entries within the core model database schema.

Model Preprocessor

The preprocessor reads--or *parses*--the files generated by the extraction process and constructs an import file which accurately models the extracted portion. The end result of completing a preprocessor is a script that is capable of accepting customer source GIS data files and generating import files.

The Model Preprocessor can be broken into individual stages called: Parse, Post Parse, Cell Explosion, Post Explode, Topology Construction, Post Topology, and Model Build Import file generation.

The following figure illustrates the stages in the preprocessor:



Parse Stage

The Parser reads the client GIS model from external files created by the Extraction process into a data structure known as an Entity Set. After this phase is completed, the resulting Entity Set will be a 'skeleton' for the complete model. The activities completed in this stage are not client specific; it will be more specific to a standard data file format (e.g., AutoCAD's DXF format, Intergraph's ISFF format, etc.). Each individual graphical object (e.g., point, line, or text) will be represented in an output file.

- **Post Parse**: Client specific processing that is used to accommodate any modification of the data that may be required prior to Cell Explosion.
- **Cell Explosion**: Cell explosion is the central phase of preprocessing. It is here that the conversion of the raw graphical objects to model objects is accomplished. The graphical objects are mapped to objects, which will appear in client's final model.
- **Post Explode**: Allows for client specific processing after Cell Explosion.
- **Topology Construction**: The inter-device connectivity for all electrical objects is constructed in this stage. The connectivity can either be explicit (i.e. 'To' and 'From' node identifiers) or based on proximity.
- **Post Topology**: The final opportunity for client specific processing.
- **Model Build Import File Generation**

Cell Explosion

The central phase of preprocessing is the conversion of graphical objects into full-fledged model objects; this conversion from a graphical object to a model object can involve a wide range of operations. These operations are specified in a text file <client>_devices.cel, which is called the explosion definition file.

The operations that may be accomplished during this phase include the following:

- **Handle Assignment -** This requires that a graphical entity be mapped to a particular class of model objects (e.g., switch, transformer, device annotation, road, water boundary, etc.) and that an index number, unique within that class, be assigned to this object.
- **Attribute Manipulation** - Attributes can be added, removed or renamed. They can also be assigned new values based upon combinations of other attribute values or the result of mathematical calculations.
- **Expansion/Replacement of One Object by Multiple Objects** - For example a transformer in the mapping system could be exploded into a transformer with a switch and a network protector.
- **Creation of Aggregate Objects** - One object may be used to represent a group of objects. For example, a recloser object may in fact represent the recloser along with a by-pass switch, a load switch, and a source switch. All of these component objects may be created and bundled into a single aggregate object during this phase.
- **Elimination of Un-Necessary Objects** - Any object not explicitly 'matched' during this phase will be eliminated; thus, this stage acts as a filter.
- **Assignment of Core Properties** - For example, phase, nominal status, NCG, and symbology can be assigned as default values for all devices.
- **Daughter Object Creation** - Creating new entities based upon information taken from an existing object.
- **Classification of Objects as Background –** Sets the location of an object to a background partition.
- **Diagnostic Messaging –** Aids in debugging or as a method to configure customer specific error messages with customer defined attributes.

Model objects have handles (class and index), attributes and aliases, geometry, and optionally aggregate object specification, all of which are supported through the explosion preprocessor.

To understand the cell definitions, which specify how an object is recognized and processed during cell explosion, one should understand two fundamental ideas:

1. "Parent" and "daughter" objects
2. String expansion.

Parent and Daughter Objects

Those objects, which enter the cell explosion process from the parser (or the post-parse processing) and which are recognized (or matched) by a definition, are considered to be "parent" objects (or, at least, potential parents); any new graphic objects created by the cell definition which matched the parent are considered "daughter" objects.

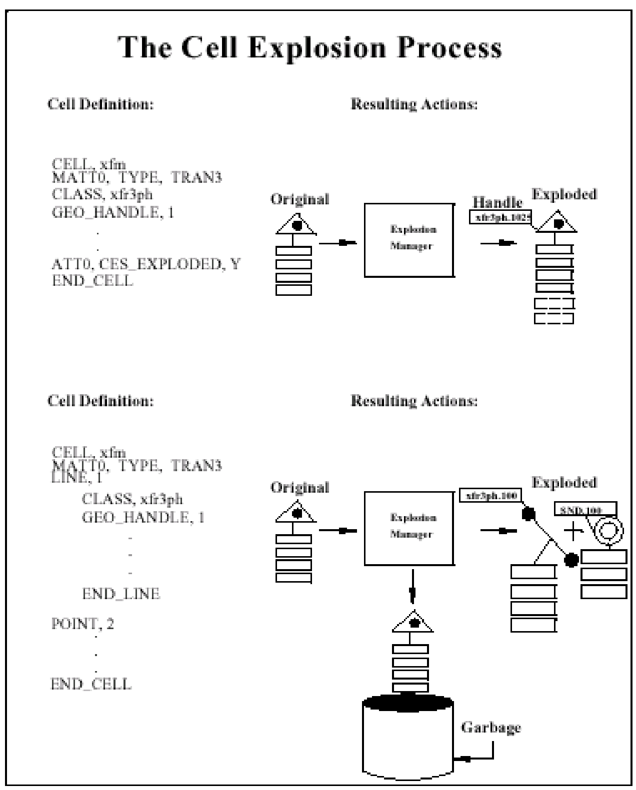
There are 4 outcomes for an object after cell explosion:

1. The parent object may pass through and be modified by cell explosion without giving rise to daughter objects.
2. The parent object may pass through cell explosion while giving rise to one or more daughter objects.
3. The parent object may be eliminated by cell explosion yet give rise to daughter objects, which survive and proceed to the succeeding stages.
4. The parent object may be eliminated by cell explosion and not give rise to daughter objects.

> **Note**: Any object that has an attribute named "CES_EXPLODED" with a value of "Y" will pass through this process; all other objects are eliminated.

Commonly, if the parent gives rise to daughter objects, the parent dies, but transfers some of its attributes to the resulting daughters through use of the ATT keyword.

The following illustration depicts outcomes 1 and 2 for an object:



String Expansion

When assigning new attributes, you may want the values for these new attributes to be formed from existing attributes--either by simply copying an existing value, or by combining and/or transforming the old values. This process is accomplished by “string expansion” which replaces or expands an attribute name into the full string representing that attribute’s value. In cell definitions, enclosing an attribute name in square brackets indicates that you intend for this attribute name to be expanded; e.g., the form “[FEEDER_ID]” will be replaced by the value of the FEEDER_ID attribute, such as “6992" (assuming that such an attribute exists for the matched object). In addition to this simple expansion, there are several specialized forms of string expansion that can be summarized as follows:

1. **Substring**

   - **Delimiter Based**

     Indicated by “<“or “>”. This this form returns the substring before or after the first occurrence of the delimiting character. The delimiting character is the character immediately following the “<“or the “>”.

     For example, if TAG= “XYZ.553”, then [<.[TAG]] returns the substring preceding the first period (“.”) in the TAG attribute value, in this case,” XYZ”. Likewise, [>.[TAG]] returns the substring following the period, which would be “553”.

     Note: When nesting a simple expansion form (e.g., [TAG]) within a delimiter based expansion form; you can discard the inner square brackets. Thus, “[<.TAG]” is equivalent to “[<.[TAG]]”.

   - **Position Based**

     Indicated by “@” -- this form returns the substring beginning and ending at the given character positions.

     Using the example from above where TAG=”XYZ.553”, the notation [@(1:2)[TAG]] extracts the substring from the value of the TAG attribute, which begins with character position 1 (position 0 being the first character) and ends with character position 2. In other words, it extracts a two-character substring, beginning from the second position, returning the value “YZ”

     Note: The character position can be specified relative to the end of the string by using the “$” character to represent the last position in the string. E.g., “[@($-1:$)[TAG]]” returns the last two characters “53”. Also note that a single character can be extracted by specifying the start and end positions as the same character, e.g., “[@(2:2)[TAG]]” returns the third character, “Z”.

2. **Codelist**

   These can be used to map or convert an input value into the corresponding output value.

   - **Basic Lookup Table:**

     To create the “lookup table”, we use the CODE keyword. The format for the table is:

     ```
     CODE, <listname>,input value, outputvalue.
     ```

     For example:

     ```
     CODE, RANK_LIST, E, 1
     CODE, RANK_LIST, R, 2
     CODE, RANK_LIST, P, 4
     ```

     creates a lookup table with three entries or mappings.

(A default code, returned when the given input value is not in the table, can be defined for a list using the DEFAULT_CODE keyword, e.g., DEFAULT_CODE, RANK_LIST, 1 means that any input value other than E, R or P results in the output of a “1”.)

To actually look up or convert a value, we use the codelist form of string expansion, indicated by a “%”.

```
[%RANK_LIST.[RANK_CODE]]
```

will return “1” if the RANK_CODE attribute is “E”; “2” if the RANK_CODE is “R”; and “4” if the RANK_CODE is “P”.

- **Database Lookup:**

  This works the same as the basic lookup table but the entries are stored in a database table. There are 2 formats for database lookups:

  - DBCODE

    The table name (which also serves as the list name), the input column name, and the output column name are defined using the DBCODE keyword. The format for the DBCODE is:

    ```
    DBCODE, <tablename>, <input column>,< output column>
    ```

    For example:

    ```
    DBCODE, feeder_ncg, feeder_name, ncg_id
    ```

    means that there exists a database table called “feeder_ncg” which has an input value column called “feeder_name” and an output value column “ncg_id”.

  - NAMED_DBCODE

    NAMED_DBCODE is similar to DBCODE except it takes a list name that is different from the table name. It is used in cases where there is a need for 2 codelists based on the same database table but with different input and output columns. The format is:

    ```
    NAMED_DBCODE, <listname>, <tablename>, <input column>,
    <output column>
    ```

  A default code, returned when the given input value is not in the table, can be defined for a list using the DEFAULT_CODE keyword. For example, DEFAULT_CODE, feeder_ncg, 1 means that any input value other than what has been defined results in the output of a “1”. Additionally, a special DEFAULT_CODE value can be assigned with the value specified as “--INTEGER_SEQUENTIAL--“.

  For example:

  ```
  DBCODE, feeder_ncg, feeder_name, ncg_id
  DEFAULT_CODE, feeder_ncg, --INTEGER_SEQUENTIAL--
  ```

  Means if a lookup into the table named feeder_ncg does not have a match, the default action will be to select the maximum value of ncg_ids in the table, add one to the ncg_id, and create a new record with the given feeder name and the incremental maximum ncg_id.

  Accessing the table is the same as the basic lookup table mentioned above.

- **Math Functions**:

  Mathematical functions can be calculated by using the input value to access a "pseudo-codelist." "List name" has one of the following values:

MATH_SIN	
MATH_COS	
MATH_TAN	
MATH_ASIN	
MATH_ACOS	
MATH_ATAN	
MATH_LOG	
MATH_LOG10	
MATH_EXP	
MATH_SQRT	
MATH_CEIL	(round up to next greatest number)
MATH_FLOOR	(round down to next lowest number)
MATH_FABS	(absolute value, e.g., -4.5 becomes 4.5)
MATH_RPN	(math function in reverse polish notation)

  For example, to calculate the sine of an ANGLE attribute:

  ```
  [%MATH_SIN.[ANGLE]]
  ```

- **Coordinate Lookup**:

  The coordinates of an object can be accessed using a form that mimics a codelist lookup:

[%COORDINATE.FIRSTX]	returns the first X coordinate of the object
[%COORDINATE.LASTX]	returns the last X coordinate of the object
[%COORDINATE.FIRSTY]	returns the first Y coordinate of the object
[%COORDINATE.LASTY]	returns the last Y coordinate of the object

3. **Default Value**

   A default value can be specified which will be returned if the result of string expansion would otherwise be an empty string. This is indicated by enclosing a default value between two caret symbols ("^").

   For example: "[^PRIMARY^[PRI_CIRCUIT_ID]]" returns a value of "PRIMARY" in any case where the PRI_CIRCUIT_ID attribute is non-existent or empty.

   If a default value is not specified, then a "String Expansion Error" message will occur.

4. **Special Attributes**

   Some properties of an object can be accessed as if they were attributes by using one of the special names given below, preceded by a double dollar sign:

CLS	(cell number)
IDX	(index number)
X1	(1st or primary X coordinate)
Y1	(1st or primary Y coordinate)
Xn	(subsequent X coordinate)
Yn	(subsequent Y coordinate)
COORD_CNT	(number of coordinates)
MAP_CLASS	(class number of partition)
MAP_NAME	(full name of partition)
CELL_NAME	(cell name, *i.e.*, the set of instructions for an object)
CLS_NAME	(actual name of class rather than number)

   For example, [$$CELL_NAME] returns the name of the "cell" within the cell definition file that was matched by the current object.

5. **Handle Reference**

   One daughter object can access the class and index number of another daughter object by using the following two forms:

   ```
   $<#>.CLS
   $<#>.IDX
   ```

   For example, in daughter object #2, the class number of daughter #1 can be accessed by the form: "[$1.CLS]" and index number of daughter #1 can be accessed by the form: "[$1.IDX]".

   > **Note**: A common practical application of this form of string expansion is to assign the DEVICE_CLS and DEVICE_IDX attributes of a SND attached to its corresponding transformer.

Available Cell Explosion Keywords

This section provides descriptions, syntax, and examples for available cell explosion keywords.

Global (outside all cell definitions)

- **CODE** - Defines an entry in a code conversion lookup table. See String Expansion Section.
- **DBCODE** - Defines an entry in a database table. See String Expansion Section.
- **DEFAULT_CODE** - Sets default values for codelist. See String Expansion Section.
- **INCLUDE** - Reads definitions from another file.

  INCLUDE, <name of file to include>

  ```
  INCLUDE,  /users/xyz/data/xyz_devices.cel
  ```

- **NAMED_DBCODE** - Allows for definitions of more than one codelist from a single database table.
- **TEMPLATE** - Uses a template definition.
- **USE** - Sets default values for an entity's properties.

  USE, <KEYWORD>, <value>

  ```
  USE, PHASE, abc
  ```

Shared (used by both parent objects & daughter graphic objects)

- **ATT[n]** - Sets the value of an attribute. There is no limit on the number of ATT[n] records that can exist in the cell definitions. [n] is currently a placeholder, usually set to 0 (zero).

  ATT[n],<att_name>, <att_value>

  ```
  ATT0, feeder, [@(9:12) [ACAD_layer]]
  ATT0, riser, N
  ```

- **ATTR_INDEX** - The string that follows this keyword will be used to assign an index unique for an object of this object's class; usually, the string will be formed by expansion of one or more attributes.

  ATTR_INDEX, <n>

  ```
  ATTR_INDEX,
  ```

- **BND_HANDLE** - Indicates that the index for this object should be provided by the boundary-node handle manager.

  ```
  BND_HANDLE, 1
  ```

- **CLASS** - Sets the class of object to explicit value.

  CLASS, <class name>

  ```
  CLASS, Xfm
  ```

- **DATT[n]** - Dynamic attribute name. [n] is currently a placeholder, usually set to 0 (zero).

  DATT[n], <att_name>, <att_value>

  ```
  DATT1,  [%LOCATION.[^0^[WITHIN_SITE_IPID]]],~
  4901.[^0^[WITHIN_SITE_IPID]]
  ```

- **GEO_HANDLE** - Indicates that a unique index should be generated based upon the object's class and geographical coordinates.

  ```
  GEO_HANDLE, 1
  ```

- **INDEX** - Sets the index of object to an explicit value.

  INDEX, <n>

  ```
  INDEX, 533
  ```

- **MARK_BGD** - Marks an object as background and sets its location to the background partition.

  ```
  MARK_BGD, 1
  ```

- **MSG[n] (or MESSAGE[n])** - Prints a message to standard output when this definition is used, where [n] is either 0, 1, 2, or 3.

  MSG[1|2|3], <message text>
  MESSAGE[1|2|3], <message text>

  ```
  MSG1, Warning:  Found stray fuse
  MSG2, Handle:  [$$CLS] . [$$IDX]
  MSG3, At (X,Y) of ([$$X1],[$$Y1])
  ```

- **NCG** - Set the entity's Network Control Group (NCG) property. (Program-style preprocessor only)

  NCG, <n>

  ```
  NCG, [@(9:12)[ACAD_layer]]
  ```

- **NOMINAL_STATE** - Sets the entity's 'NOMINAL_STATE' property. The value can be an integer typically between 0 and 15 or the key words OPEN or CLOSED.

  NOMINAL_STATE, <n>|OPEN|CLOSED

  ```
  NOMINAL_STATE, CLOSED
  ```

- **OPT_ATT[n]** - Sets an optional value of an attribute. Will not report a string error message if the value fails on attribute expansion. [n] is currently a placeholder, usually set to 0 (zero).

  OPT_ATT[n], <att_name>, <att_value>

  ```
  OPT_ATT1, From_Node_Bnd, [NODE1_BND]
  ```

- **OPT_DATT[n] -** Sets an optional dynamic attribute name. Will not report a string error message if the attribute name fails on attribute expansion. [n] is currently a placeholder, usually set to 0 (zero).

  OPT_DATT[n]

  ```
  OPT_DATT1, [%LOCATION.[^0^[WITHIN_SITE_IPID]]],~
   4901.[^0^[WITHIN_SITE_IPID]]
  ```

- **PHASE** - Sets the entity's 'PHASE' property (e.g., to ABC).

  PHASE, <n>

  ```
  PHASE, [%PHASE_LIST.[@(6:8)[ACAD_layer]]]
  ```

- **STRING** - Sets the value of the text string for this entity. (TEXT objects only)

  STRING, <string>

  STRING, [KVAR]

- **SUB_BND** - Indicates that this object is a substation boundary node and that its index should be assigned based upon the supplied string (usually the feeder or circuit identifier).

  ```
  SUB_BND, 1
  ```

- **SYM_ID** - Sets the symbology-state-class to an explicit value, rather than its default value, which is the same as the class number.

  SYM_ID, <n>

  ```
  SYM_ID, 1304
  ```

- **VOLTS** - Sets the entity's 'VOLTS' property. (Program-style preprocessor only)

  VOLTS, <n>

  ```
  VOLTS, [voltage] 1000 *
  ```

Parent Object ("explosionDef") Only

- **AGGREGATE/_ POINT/_LINE/_TEXT** - Creates a graphic object of the specified kind that becomes a component of the overall aggregate device. AGGREGATE and AGGREGATE_LINE require 2 coordinates; AGGREGATE_POINT and AGGREGATE_TEXT require one coordinate. All AGGREGATE definition types require an END_AGGREGATE. (Obsolete)

  AGGREGATE, <n>

  ```
  AGGREGATE, 4
  ```

  AGGREGATE_ POINT, <n>

  ```
  AGGREGATE_POINT, 1
  ```

  AGGREGATE_ LINE, <n>

  ```
  AGGREGATE_ LINE, 3
  ```

  AGGREGATE_ TEXT, <n>

  ```
  AGGREGATE_ TEXT, 2
  ```

- **CELL** - Begins the definition for one device type. The cell definition file can contain many sets of cell definitions. All CELL definitions require an END_CELL.

  CELL, <name>

  ```
  CELL, uxfm2
  ```

- **END_CELL** - Ends an explosion definition.

  ```
  END_CELL
  ```

- **END_AGGREGATE** - Ends an aggregate definition.

  ```
  END_AGGREGATE
  ```

- **END_TEMP** – Ends a template definition.

  ```
  END_TEMP
  ```

- **MATT[n]** - Matching attribute of the object to explode. [n] is currently a placeholder, usually set to 0 (zero). There is no limit on the number of MATT[n] records a cell explosion definition may have, but for the explosion to occur, all must match.

  MATT[n],<attribute name>,<target attribute value>

  ```
  MATT0, ACAD_objectType, INSERT
  ```

- **POINT/LINE/TEXT** - Creates a "daughter" graphic object of the specified kind. All POINT/LINE/TEXT definitions require an END_POINT/LINE/TEXT.

  POINT, <n>

  POINT, 3

  LINE, <n>

  LINE, 1

  TEXT, <n>

  TEXT, 5

- **POINT/LINE/TEXT WHEN <condition>** - Creates a "daughter" graphic object of the specified kind when the given condition is met. All POINT/LINE/TEXT definitions require an END_POINT/LINE/TEXT.

  POINT WHEN <condition>

  POINT WHEN

  LINE WHEN <condition>

  LINE WHEN

  TEXT WHEN <condition>

  TEXT WHEN

- **POINT/LINE/TEXT FOR** <*variable*> **IN** <*List of Values*> - Creates zero, one or multiple graphic objects of the specified kind, one object for each value in the supplied list. Use <*variable*> within the definition as if it were an attribute name. A special variable called "$$ICOUNT" can also be used to retrieve the number of the iteration. All POINT/LINE/TEXT definitions require an END_POINT/LINE/TEXT.

  POINT FOR <variable> IN <list of values>

  POINT FOR

  LINE FOR <variable> IN <list of values>

  LINE FOR

  TEXT FOR <variable> IN <list of values>

  TEXT FOR

- **POINT/LINE/TEXT FOR** <*num-valu***e> TIMES** - Creates zero, one or multiple graphic objects of the specified kind; number of objects specified by <*num-value*s>. ($$ICOUNT can be used just as for the previous form). All POINT/LINE/TEXT definitions require an END_POINT/LINE/TEXT.

  POINT FOR <numeric value> TIMES

  POINT FOR

  LINE FOR <numeric value> TIMES

  LINE FOR

  TEXT FOR <numeric value> TIMES

  TEXT FOR

- **REQUEST_HANDLE** - Indicates that the existing handle of this object should be replaced with one supplied by the Explosion manager's "ExplodeHandle" class. (Primarily for ISFF)

  ```
  REQUEST_HANDLE, 1
  ```

- **RMV[n]** - Removes an attribute. [n] is currently a placeholder, usually set to 0 (zero).

  RMV[n]

  ```
  RMV0, voltage
  ```

- **RNA[n]** - Renames an attribute. [n] is currently a placeholder, usually set to 0 (zero).

  RNA[n], <att_name>, <new_att_name>

  ```
  RNA0, amp_content, amp_cont
  ```

- **TEXT_SCALE** - Specifies the scale factor for text. Used to allow the height of base text symbol to be used as a multiplier to the cell definition specified coordinates.

  TEXT_SCALE, <n>

  ```
  TEXT_SCALE, 1
  ```

  for example with the TEXT_SCALE, 1 specified and the base text object has a specified height of 400 and the COORD1, 10, 30 is specified, the resulting coordinates will be 400x10, 400x30 or 4000, 12000.

- **USE_REFERENCE** - Indicates that the index for this object should be based upon its corresponding reference object. (ISFF only) (Obsolete). For example:

  ```
  USE_REFERENCE, 1
  ```

  causes the FRAMME RB_REFPRMRY and RB_REFSCNDRY linkages to be used instead of the normal RB_PRIMRY and RB_SECNDRY.

Component "Daughter" Object ("explosionGrObject") Only

- **ABSOLUTE_COORDS** - Indicates that coordinate values are specified in absolute, "real-world" numbers; this over-rides the default behavior which is for numbers used in COORD statements to be taken as relative to the insertion point of the parent object (i.e. this insertion point corresponds to COORD 0.0, 0.0).

  ```
  ABSOLUTE_COORDS, 1
  ```

- **ANGLE** - Sets the text rotation for this entity. Horizontal is zero and the angle proceeds counter clockwise. (TEXT objects only)

  ANGLE, <a>

  ```
  ANGLE, 90
  ```

- **COORD/COORD[n]**- Sets relative/absolute coordinate of an object/endpoint.

  COORD, <x>, <y>

  ```
  COORD, 1.0, 2.5
  ```

  COORD[n], <x>, <y>

  ```
  COORD1, 0.0, 1.0
  COORD2, 1.0, 2.0
  ```

- **COMPONENT[n]** - Sets the aggregate sequence number and cell component number for a single component in the aggregate.

  COMPONENT[n], <agg_seq_num>, <cell_comp_num>

  ```
  COMPONENT1, 1, 2
  ```

- **END_AGGREGATE** - Ends the definition of component graphic object.

  ```
  END_AGGREGATE
  ```

- **HEIGHT** - Sets the text height for this entity. (TEXT objects only)

  HEIGHT, <h>

  ```
  HEIGHT, 2
  ```

- **H_ORIENTATION** - Sets the horizontal justification of text. Values can be LEFT, CENTER, or RIGHT, or 0, 1, or 2. Default is LEFT. (TEXT objects only)

  H_ORIENTATION, <n>|LEFT|CENTER|RIGHT

  ```
  H_ORIENTATION, LEFT
  ```

- **USE_ROTATION** - Indicates that the rotation property of the original entity should be used to set the rotation for the component graphic object.

  ```
  USE_ROTATION, 1
  ```

- **V_ORIENTATION** - Sets the vertical justification of text. Values can be TOP, CENTER, or BOTTOM, or 0,1, or 2. Default is BOTTOM. (TEXT objects only)

  V_ORIENTATION, <n>|TOP|CENTER|BOTTOM

  ```
  V_ORIENTATION, 2
  ```

Special Attributes Set by Explode an Processed by mat2entityset.(script-preprocessor):

- **Alias -** Sets an alias for an attribute (both script- and program-style preprocessors)**.**

  ATT[n], ALIAS[dbtype], <value>

  ```
  ATT0, ALIAS[OPS], [LOC_NUM]
  ```

- **Diagram-id** - Sets the Diagram Id **.**

  ATT[n], DIAGRAM_ID, <value>

  ```
  ATT1, DIAGRAM_ID, [IPID]
  ```

- **Group –** Sets the Group code**.**

  ATT[n], CES_PP_GROUP|GROUP|Group|group, <value>

- **Local**

  ATT[n], LOCAL|Local|local, <value>

- **Locations** (not to be confused with LOCATIONS)

  ATTN[n],CES_LOCATION, <value>

  ```
  ATT1,  CES_LOCATION, 4901.[MID]
  ```

  ATTN[n], CES_LOCATION_DEFINITION, <value>

  ```
  ATT1,  CES_LOCATION_DEFINITION, 4901.[MID]
  ```

  ATT[n], CES_LOCATION_NAME, <value>

  ```
  ATT1,  CES_LOCATION_NAME, Pole [^?^[SUPPORT_NO]]
  ```

  ATT[n], CES_LOCATION_DESC, <value>

  ```
  ATT1,  CES_LOCATION_DESC, Pole defined by support/switch:~
     [^?^[SUPPORT_NO]]/[^?^[SWITCH_NAME]]
  ```

  ATT[n], CES_LOCATION_REFERENCE, <value>

  ```
  ATT1,  CES_LOCATION_REFERENCE, [%COORDINATE.FIRSTX],~
  [%COORDINATE.FIRSTY]
  ```

Network Control Group

ATTN[n], NCG|Ncg|ncg, <value>

```
ATT1,  NCG, [%feeder_ncg.[^UNKNOWN^[DISTRICT]]_ ~
[%ncg_volt.[^UNKNOWN^[VOLT_LEV]]]]
```

- **Rank**

  ATT[n], RANK|Rank|rank, <value>

  ```
  ATT1,  RANK, [%MATH_RPN.[%RANKU.[^NO^[URBAN]]]~
  [%RANKLC.[^UNKNOWN^[LINE_CATEGORY]]] + ~
  [%RANKV.[^0^[VOLT_LEV]] [^0^[VOLT_LEV]]] + ~
  [%RANKB11.[^0^[VOLT_LEV]] [^UNKNOWN^[DISTRICT]]] + ~
  [%RANKP.[^RYB^[PHASING]]] +]
  ```

- **Physical Property**

  ATT[n], CES_PHYS_PROP|PHYS_PROP|Phys_Prop|phys_prop|physical_property, <value>

  ```
  ATT0, CES_PHYS_PROP, [%MATH_RPN.[%PHYS_PROP.BACKBONE] [%PHYS_PROP.~
  [^OH^[OH_UG]]] +]
  ```

- **Topology specific**

  ATT[n], From_Node, <value>

  ```
  ATT1,  From_Node, [FROM_NODE]
  ```

  ATT[n], To_Node, <value>

  ```
  ATT1,  To_Node, [TO_NODE]
  ```

  ATT[n], Unique_id, <value>

  ```
  ATT1,  Unique_Id, [FROM_NODE]_[TO_NODE]_FID
  ```

- **Transition**

  ATT[n], TRANSITION_ID|Transition_ID|Transition_Id|transition_id, <value>

  ```
  ATT1,  TRANSITION_ID, 120
  ```

- **Voltage**

  ATT[n], VOLTAGE|Voltage|voltage, <value>

  ```
  ATT1,  VOLTAGE, [%VOLTS.[^UNKNOWN^[OPERATING_VOLTAGE]]]
  ```

Format for the Explosion Definition File

Devices are recognized, or 'matched', and appropriate manipulations are made based upon the descriptions or definitions contained in an explosion definition text file.

The general format for a single cell definition is as follows:

```
CELL, <cell-name>
       <match-criteria>
       [ <parent-object-actions> ]
       [ <daughter-object-actions> ]
END_CELL
```

Remember, any object that has an attribute named "CES_EXPLODED" with a value of "Y" will pass through the explosion process (ATTO, CES_EXPLODE, Y); all other objects are eliminated.

Syntax

Cell Definition

1. One statement per line (the ~ can be used to continue on more than one line).
2. Comments begin with # and must be on a line by themselves.
3. Lines begin with keywords (always upper case).
4. Commas separate keywords and values.

   Value fields can be:

   - Attribute substituted using the syntax [<att name>] where the value of the <att name> for the currently exploded object will be substituted in the value string. See the examples in the line definition above.
   - Math functions in Reverse Polish Notation (RPN) with space delimitation. The keywords which support RPN automatically are:
     - ANGLE
     - HEIGHT
     - H_ORIENTATION
     - INDEX
     - NCG
     - NOMINAL_STATE
     - SYMBOLOGY
     - VOLTS
     - V_ORIENTATION

     For example, the following will be valid:

     ```
     COORD, 100.0, 300.0
     COORD, 100.0 [X_OFFSET] +, 300.0 [Y_OFFSET] +
     COORD, [X_OFFSET], [Y_OFFSET]
     ```

   Math operators supported include +, -, *, /, % (modulus) and ^ (exponentiation).

During the Parse phase of the preprocessor, the customer's raw data files are converted into an internal data structure known as an Entity Set wherein each individual graphical object is represented by an Entity object. Each Entity object is read into the cell file and is processed separately. When creating a cell definition file, to decrease processing time:

1. Place filter cells at the top of the file. For example, cells with nothing but match criteria that will not be exploded.
2. Place cells with most abundant objects near the top of the file. For example, if a file contains 20 switches, 10,000 text objects and 500 transformers, place the text objects first, transformers next, and finally the switches.
3. Place most restrictive criteria cells for objects above general. Overhead transformers should be placed above generic transformers in the cell definition file.

Match Criteria

1. Use keyword MATT[n].
2. Basic form: MATT[n],<attribute name>,<target attribute value>.
3. Attribute name can be replaced by a string expansion.
4. Can use alternation of target values separated by `|`.

   ```
   MATTO,[ACAD_layer],15kv-Bus|24kv-Bus|161kv-Bus
   ```
5. Multiple match criteria are logically "AND" ed together. All MATT[n] must return true before that cell will be used. For example, for the following cell to be used for an Entity object, all 3 lines must return true:

   ```
   CELL, 01XF1
      MATTO, ACAD_objectType, INSERT
      MATTO, ACAD_blockName, 01XF1
      MATTO, [@(1:3) [ACAD_layer]], PRI
      …
   ```

Conditional Expressions

These have the form:

```
( (Boolean-Expression) ? true value | false value )
   ATT0, ALIAS[OPS], ( ([location]) ?  [location]|D:[ATTR] )
```

The supported syntax for Boolean expressions within cell-definition files is as follows:

```
<Expression> = <Expression> && <Expression>
   <Expression> || <Expression>
   !<Expression>
   (Expression)
   <String-Comparison>
   <Numeric-Comparison>
   <Term>
```

where

```
<String-Comparison> = <String> == <String>
   <String> != <String>
   <String> < <String>
   <String> > <String>
   <String> <= <String>
   <String> >= <String>
```

where

```
<Numeric-Comparison> = <Number> .eq. <Number>
   <Number> .ne. <Number>
   <Number> .lt. <Number>
   <Number> .le. <Number>
   <Number> .gt. <Number>
   <Number> .ge. <Number>
```

where

```
<Term> = <String> | <Number> | <Function-Call>
```

where

```
<String> = <Simple-String> | <Expand-Form>
```

where

`<Simple-String>` = double-quoted string of alphanumeric characters (*e.g.*, "553").

`<Expand-Form>` = attribute or property name enclosed in square brackets (*e.g.*, [att_name])

`<Number>`

`<Function-Call>` = name of a standard function with argument(s) enclosed in matched parentheses.

Note: At present no standard functions have been implemented, so this feature should not be used.)

Operators are evaluated in the following order, with top most operators processed first. The operators used are:

```
!
< > <= > >= .lt. .gt. .le. .ge.
== != .eq. .ne.
&&
||
```

Examples:

```
([Layer] .eq. 501)
(([ObjectType] != "Primary Conductor") && ( [FeederId] .ne. 6800 ))
( sin(Rotation) < 0.5 )
( ![UniqueId] )
```

Example of Cell Definitions

Tranformer w/Supply Node

```
CELL, OverheadTransformer
  MATT0, CESMP_OBJ_CLASS, Transformer
  MATT0, [OhUg], OH
  MATT0, DIAGRAM_ID, Symbol

  LINE, 1
    ABSOLUTE_COORDS,1
    COORD1,[$$X1],[$$Y1]
    COORD2,[$$Xn],[$$Yn]

    # Definition attributes
    CLASS, xfm_oh
    SYM_ID, 2060[%phase_num.[^ABC^[Phase]]]
    ATTR_INDEX, [GUID]
    ATT0, ALIAS[OPS], [DeviceId]
    ATT0, ALIAS[GIS], [GisId]
    NCG, [%feeder_ncg.[CESMP_MAPNAME]]
    ATT0, NCG_FDR, [CESMP_MAPNAME]

    # Topology definition
    PHASE, [%phase_map.[^ABC^[Phase]]]
    NOMINAL_STATE, [%status_lookup.[^CLOSED^[NominalStatus]]]
    VOLTS, [%voltage.[^4160^[Voltage]]]
    PHY_PROPERTIES, [ces_physical_property]
    ATT0, From_Node, [_Connector0])
    ATT0, To_Node, [_Connector0]_SND

    RANK, [%phase_bit.[^ABC^[Phase]]]
 [%voltage_bit.[%voltage.[^4160^[Voltage]]]] +
    # Attribute mapping
    OPT_ATT0, facility_id,             [GisId]
    OPT_ATT0, device_name,             [DeviceId]
    OPT_ATT0, feeder_id_1,             [FeederName]
    OPT_ATT0, feeder_id_2,             [FeederName2]
    # Explode this object
    ATT0, CES_EXPLODED, Y
  END_LINE
    POINT, 6
      CLASS, SND
      ATTR_INDEX, [GUID]
      PHASE, [%phase_map.[Phase]]
      SYM_ID, 994
      NCG, [%feeder_ncg.[CESMP_MAPNAME]]
      COORD, 0, -1
      ATT0, Unique_Id, [_Connector0]_SND
      ATT0, device_cls, [$1.CLS]
      ATT0, device_idx, [$1.IDX]
      ATT0, device_id, [DeviceId]
      ATT0, feeder, [$$MAP_NAME]
      ATT0, phases, [%phase_num.[Phase]]
      ATT0, ncg, [%feeder_ncg.[CESMP_MAPNAME]]
      ATT0, CES_EXPLODED, Y
    END_POINT

END_CELL
```

Code Lookup Exampes

Below is an example of how a lookup table can be used to convert the GIS phase to a NMS phase:

```
#
# CODE phase_map
#
CODE, phase_map, 1, A
CODE, phase_map, 2, B
CODE, phase_map, 4, C
CODE, phase_map, 3, AB
CODE, phase_map, 5, AC
CODE, phase_map, 6, BC
CODE, phase_map, 7, ABC
CODE, phase_map, A, A
CODE, phase_map, B, B
CODE, phase_map, C, C
CODE, phase_map, AB, AB
CODE, phase_map, BA, AB
CODE, phase_map, AC, AC
CODE, phase_map, CA, AC
CODE, phase_map, BC, BC
CODE, phase_map, CB, BC
CODE, phase_map, ABC, ABC
CODE, phase_map, CBA, ABC
CODE, phase_map, BCA, ABC
CODE, phase_map, BAC, ABC
CODE, phase_map, CAB, ABC
CODE, phase_map, Unknown, ABC
CODE, phase_map, Null, ABC
DEFAULT_CODE, phase_map, ABC
```

Below is an example of using a lookup table (a.k.a. codelist) that is stored in a database table.

```
#
# CODE feeder_ncg
#
DBCODE, feeder_ncg, feeder_name, ncg_id
DEFAULT_CODE, feeder_ncg,  --INTEGER_SEQUENTIAL--
```

Below is an example of using a single lookup table (a.k.a. codelist) that is stored in a database table where you need multiple fields returned.

```
#
# CODE pf_capacitor_data_kvar_rating_a
#
NAMED_DBCODE, pf_capacitor_data_kvar_rating_a, pf_capacitor_data,
  catalog_id, kvar_rating_a
DEFAULT_CODE, pf_capacitor_data_kvar_rating_a, 0

#
# CODE pf_capacitor_data_kvar_rating_b
#
NAMED_DBCODE, pf_capacitor_data_kvar_rating_b, pf_capacitor_data,
  catalog_id, kvar_rating_b
DEFAULT_CODE, pf_capacitor_data_kvar_rating_b, 0

#
# CODE pf_capacitor_data_kvar_rating_c
#
NAMED_DBCODE, pf_capacitor_data_kvar_rating_c, pf_capacitor_data,
catalog_id, kvar_rating_c
DEFAULT_CODE, pf_capacitor_data_kvar_rating_c, 0
```

Model Build Workbooks

The core model preprocessor configuration files are maintained and generated from the two model build workbooks, the NMS_System_Distribution_Model workbook and the Oracle Utilities Network Management System Power Engineering Woorkbook.

System Distribution Model Workbook

The modeling workbook contains many tabs to map a customer's GIS data to the standard NMS model. These tabs include device-mapping tabs, attribute-mapping tabs, and a "Tools" tab containing tools used to automate model and preprocessor configuration. Mapping is accomplished by assigning each GIS object an NMS class based on specified criteria. Attributes associated with the GIS objects mapped are then also mapped to NMS attributes in their appropriate attributes tab. The mapping information entered into these tabs will be used to generate a set of customer specific model and preprocessor configuration files.

The System Distribution Model workbook maintains and generates the following model configuration files:

- Classes File
- Inheritance File
- Attribute Schema File
- Attribute Configuration File
- State Mapping File
- Voltage Symbology File
- Rank Configuration File
- Hide/Display File
- Declutter File
- Electrical Layer Objects File
- Landbase Layer Objects File

Model Configuration Files Generated by the Workbook

The modeling workbook is a tool used to generate model and preprocessor configuration files. Below is a list of all the files generated by the workbook with a brief description. Notice that <project> indicates that the files generated pertain to a specific project configuration.

File	Description
<project>_classes.dat	Contains all NMS classes being used in the current workbook mapping.
<project>_inheritance.dat	Contains the inheritance structure of all classes being used in the current workbook mapping. This structure may include NMS required inheritance definitions.
<project>_schema_attributes.sql	Contains the schema definition for all attributes in the NMS Model. Along with the schema definition, a view is also defined for each database table created. The view is created based on the display names provided in the attribute tabs.

File	Description
<project>_attributes.sql	Contains the attribute mapping specified in each of the attribute tabs. This mapping is used during model build time to insert the specified attribute mapping into the appropriate NMS model tables.
<project>_ssm.sql	Contains a symbol to device mapping based on the nominal and current states of the device.
<project>_devices.cel	Contains the actual mapping criteria definition for all electrical devices. The criteria are derived from the information in the mapping tabs.
<project>_landbase.cel	Contains the actual mapping criteria definition for all landbase objects.

Mapping Tabs

There are ten object-mapping tabs in the workbook. These tabs are used to specify the GIS object and the exact criteria for a GIS object to map to the selected NMS class. Below is a list of all the mapping tabs with a brief description.

Workbook Tab	Description
Core Nodes	This tab contains all NMS core nodes. These core nodes are used during CELL file generation. They will not be included in the classes and inheritance files.
Devices	Intended for the mapping definition/criteria of all electrical objects (Switches, Transformers and other operable devices).
Conductors	Intended for the mapping definition/criteria of all conductor objects.
Customer & Service	Intended for the mapping definition/criteria of all electrical service devices. Such as point of service, generators and meters.
Structures	Intended for the mapping of structure objects, such as manholes, poles and switchgear cabinets.
Landbase	Intended for mapping of all background parcel data.
Annotation	Used to map text objects from both the electrical and background layers to specific SPL classes.
Gas Devices	
Gas Pipes	
Gas Annotation	

Mapping Syntax

To take advantage of the tools included in the workbook, the correct syntax must be used. The workbook is to be mapped using a simpler syntax than the CELL explosion language. When in doubt about specific syntax, you can always assume that if it conforms to the CELL explosion language, it will work for the workbook mapping.

Class Mapping Columns and Syntax

Column	Description
Parent Class	This is a locked column and should only be modified by NMS model engineers. This column is used to define the inheritance lattice. The class in this column defines the parent for the child found in the next column "Class Name". Multiple parents can be defined for a single class using a comma "," to separate the class names.
Class Name	This is a locked column and should only be modified by NMS model engineers. This column indicates the name of the class.
Attribute Table	This is a locked column and should only be modified by NMS model engineers. This column indicates the table in which the attributes associated with this class will be stored.
Class Number	This is a locked column and should only be modified by NMS model engineers. The number in this column indicates the class number of the NMS class.
Index	This column is used to specify the index to be used during CELL file generation. The syntax for this column is CELL explosion language syntax. The CELL file generated will always use attribute index (ATTR_INDEX) to specify an index for a specific object using the data found in this column. Example: [ATT_TransformerOH.OBJECTID]
Phase	The criteria specified in this column will be used during CELL file generation to specify a phase value to the device being processed. If this column is left blank, ABC phase will be used. Example: [ATT_TransformerOH.PHASES]
Nominal Status	The criteria specified in this column will be used during CELL file generation to specify the nominal status of the device as it is being processed. If this column is left blank, CLOSED will be used. Example: [ATT_TransformerOH.NORMALLSTATE]
NCG	The criteria in this column will be used during CELL file generation to indicate the network control group of the device being processed. Example: [%feeder_ncg.[ATT_TransformerOH.[CIRCUITID]]
From_Node	The criteria in this column will be used during CELL file generation to indicate the topological from connection. Example: [OBJ_PORT_A]
To_Node	The criteria in this column will be used during CELL file generation to indicate the topological to connection. Example: [OBJ_PORT_B]
Physical Properties	The criteria in this column will be used during CELL file generation to specify the special characteristics of this device such as lateral or backbone. Example: [%phys_prop.[ATT_TransformerOH.PROPERTIES]]
Rank	The criteria in this column will be used during CELL file generation to specify the rank to be used for hide display configuration. Example: [%rank_bit_mask.[OBJ_CLASS]]

Column	Description
Capable Phases	The value in this pull down menu will be used during state mapping generation. It is used to indicate the possible phases a device can have. This information is important when generating the permutations needed for symbol mapping.
Gang Operated	The value in this pull down menu will be used during the generation of the inheritance lattice. If gang operated is selected, the class it is set for will contain an additional parent of “gang_operated”.
Outage Stop Class	This value is not currently being used.
Symbology Enumerator	The criteria in this column will be used during CELL file generation to specify the symbology ID for the device. Example: 1050[%phs_num.[ATT_TransfomerOH.PHASES]]
Coordinate Definition	The criteria in this column will be used during CELL file generation. The CELL file generated will always use relative coordinates. If absolute coordinates are require, then the ABSOLUTE_COORDS, 1 key word must be specified. If this column is not populated then the following will be used: COORD1, 0, 0 COORD2, 0, 10 Example: ABSOLUTE_COORDS, 1 COORD1, [ATT_X1], [ATT_Y2] COORD2, [ATT_X2], [ATT_Y2]
Add Text Mapping	The values in this column should only be added through the text-mapping window. The window starts by clicking on the column button (“Add Text Mapping”). Specify the row and column for the class the mapping is intended for. All information in the form is to be entered using CELL file syntax. The information entered for the text class mapped will be saved to the tab “Text Mapping”. Multiple text classes can be added for each class. When a text class is mapped and saved from the text-mapping window, the text class used will be populated in the “Add Text Mapping” column.
Alias Definition	The criteria in this column will be used during CELL file generation. Example: SW-[%sw_type.[ATT_Switch.FACILITY_TYPE]]
Display Name	The value in this column must be unique to the workbook and must not contain any spaces. This value is used as the display name for the control tool title.
GIS Object	The criteria in this column indicate the GIS object or feature class that will be used during the mapping in the CELL file (Example: MATT0, [ATT_TYPE], SWITCH). Multiple objects or GIS features can be separated by the “|” (OR) identifier. Example: SubstationDevices|CircuitBreaker

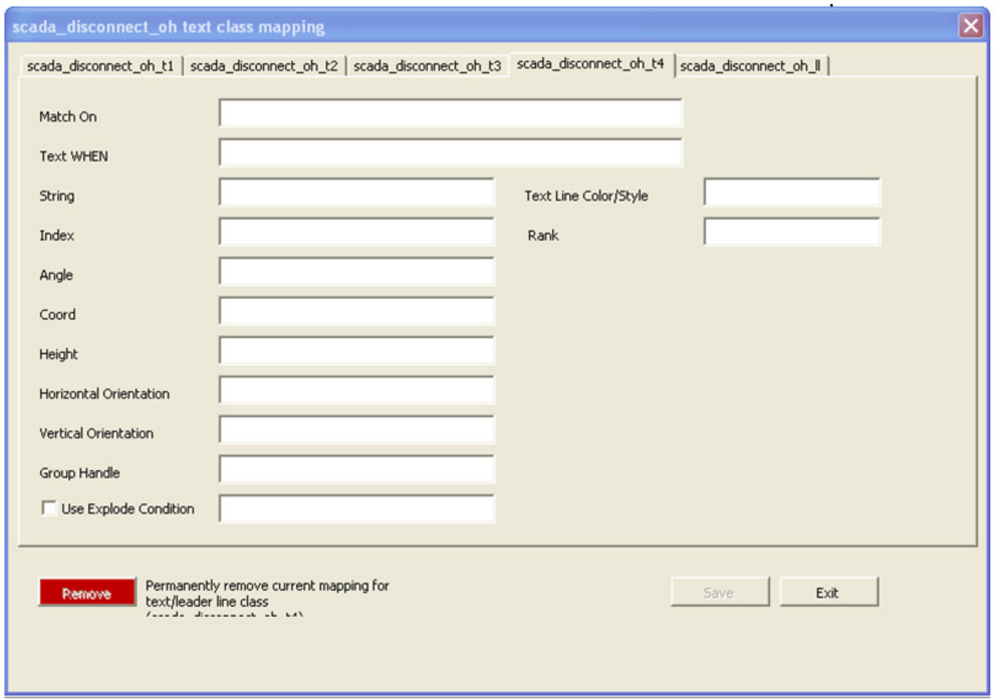
Column	Description
GIS Attribute that qualifies extraction	The criteria in this column indicate the GIS attribute to test on during the mapping stage. Multiple attributes can be used. Multiple attributes will be “AND” ed together. To indicate that multiple attributes are to be tested, a new line must separate the attributes. The OR condition cannot be used. Example: (AND) SubstationDevices.SUBTYPE SubstationDevices.SCADACONTROLLED
GIS Attribute criteria for extraction	The criteria in this column indicate the GIS attribute value that must be found for the expression to be true. Multiple values can be listed in an OR condition separated by the “|” character. For an AND condition, the values must be separated using a new line. The amount of new lines must match the number of new lines in the previous column. Example: CircuitBreaker SCADA Controlled
Comments	This column is intended for any additional comments desired to better inform the customer or model engineer of what is desired.
MP File Object	This column is not required. It is intended to provide more information about the object definition as found in the MP file.
MP Qualifying Attributes	This column is not required. It is intended to provide more information about the attribute names as found in the MP file.
Special Processing	This column is used to indicate that special processing exists for a particular device mapping. The “Special Processing” tab should be populated with the special CELL file criteria to be added to the mapping. The “Display Name” column is used to indicate the link to the “Special Processing” tab.
Comments	This column is intended for any additional comments desired to better inform the customer or model engineer of what is desired.

Attribute Mapping Columns and Syntax

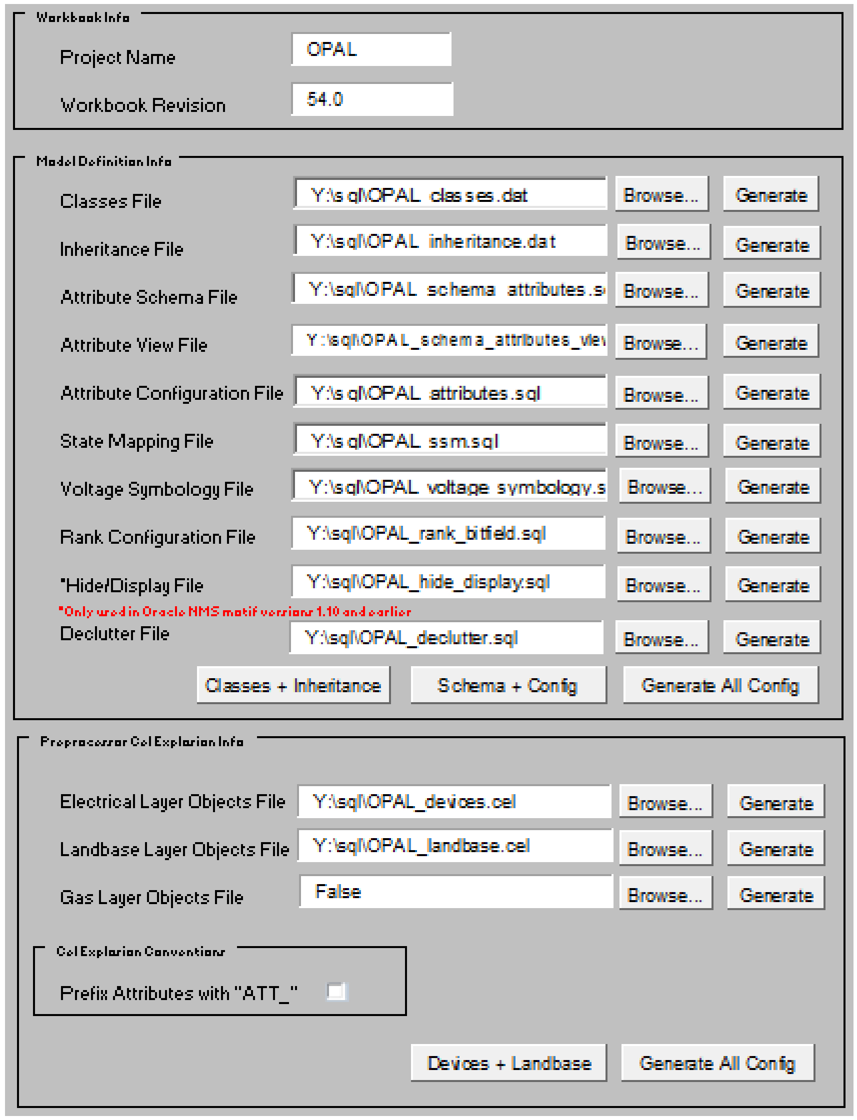
Column	Description
Attribute	The NMS model attributes being mapped. This column is locked and should not be modified.
Example Value	Example information, where appropriated. This column is locked.
Data Type	The data type of the attribute being mapped. This column is locked and should only be changed by an NMS model engineer.
Required / Recommended	Indicates if this attribute is required or recommended and indicates by which module the attribute is required or recommended. The color is used to indicate if it is required or recommended.
Field Order	Not currently used.
Display Name	Specifies the name of the attribute, as it will be displayed in the Attribute Viewer. If one display name is set, it assumes all attributes will have a display name and uses the NMS attribute name if no display name is specified. Only attributes containing values will be displayed in the Attribute Viewer Tool.
GIS Class	Indicates the name of the GIS object or feature.
GIS Attribute	Indicates the name of the GIS attribute. This column is critical to correct attribute mapping in the CELL file. The prefix of ATT_ is not required for script style preprocessor as long as the “Use ATT_ Prefix” is selected in the “Tools” tab. Complex mapping should be done using lookups and/or conditional statements in CELL file syntax.
Comment	Used to specify additional information that may be useful to the modeler or customer.
MP File Objects	This column is not required. It is intended to provide additional information about the object as found in the MP file.
MP Qualifying Attributes	This column is not required. It is intended to provide additional information about the attribute as found in the MP file.
Special Processing	This column is not required.
Comment	Used to specify additional information that may be useful to the modeler or customer.

Text Mapping Window

The text-mapping window is to be used for text mapping when the text to be displayed is not included in the data as a separate object. This is true for most attribute based annotation GIS systems. The screen capture below is an example of how a text object can be created for a device class based on the value of an attribute.



Generation Tools

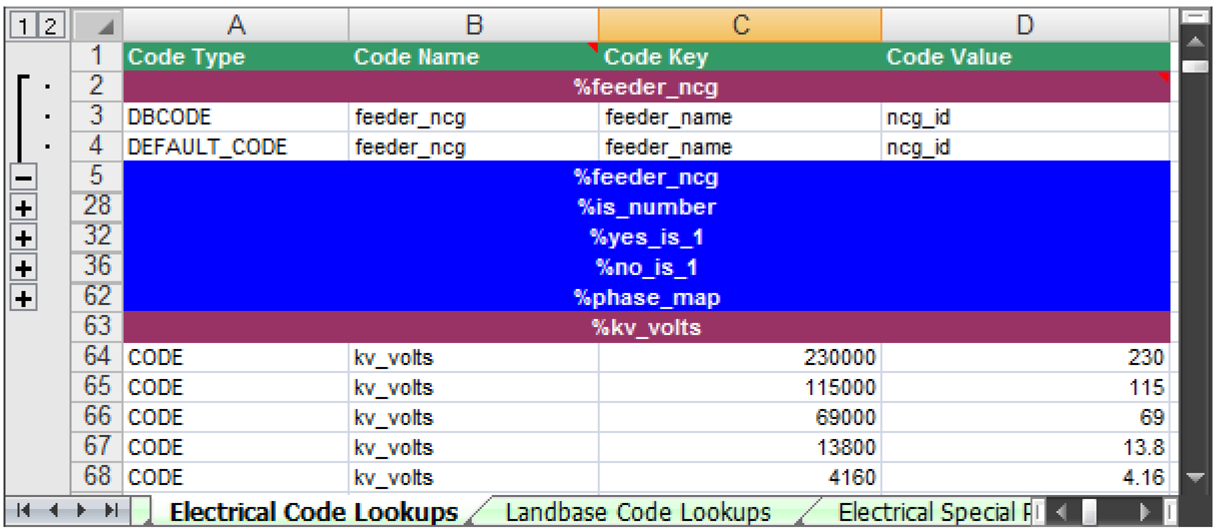


Code Lookups

All code lookups to be used in the mapping of the workbook must be specified in their appropriate tab in the workbook. This information is to be entered by the NMS model engineer. Lookups can be database lookups by specifying them as db code lookups in the appropriate CELL file syntax.

Electrical Code Lookups	Contains lookups to be included in the Electrical Layer Objects Cell File.
Landbase Code Lookups	Contains lookups to be included in the Landbase Layer Objects Cell File.
Gas Code Lookups	Contains lookups to be included in the Gas Layer Objects Cell File.

Code Lookups Example



	A	B	C	D
1	Code Type	Code Name	Code Key	Code Value
2	%feeder_ncg			
3	DBCODE	feeder_ncg	feeder_name	ncg_id
4	DEFAULT_CODE	feeder_ncg	feeder_name	ncg_id
5	%feeder_ncg			
28	%is_number			
32	%yes_is_1			
36	%no_is_1			
62	%phase_map			
63	%kv_volts			
64	CODE	kv_volts	230000	230
65	CODE	kv_volts	115000	115
66	CODE	kv_volts	69000	69
67	CODE	kv_volts	13800	13.8
68	CODE	kv_volts	4160	4.16

Electrical Code Lookups | Landbase Code Lookups | Electrical Special F

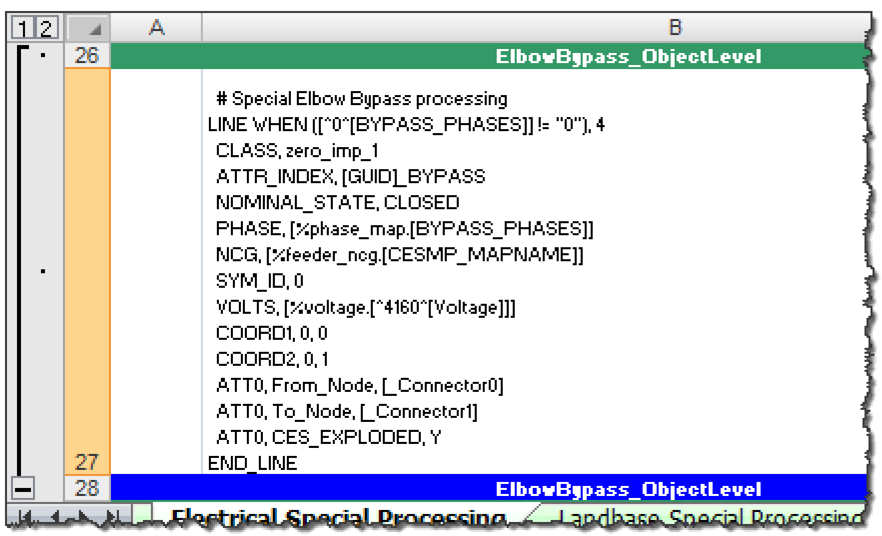
Special Processing Tabs

The Special Processing tabs are arranged according to the cell file they should be included in. A model engineer can use these tabs to add any special CELL file enhancement that cannot be fully generated by the workbook. This includes the addition of nodes such as FBD, FID, SRC, and SND nodes. There are two hooks for each CELL file block generated. One is at the device level, before the end of the first object's END_LINE or END_POINT). The second is before the cellblock is over, before the END_CELL.

To specify that special processing is required, populate the Special Processing tab in the appropriate class-mapping tab with the display name of the class that requires special processing. The special processing to be used must be specified in a single cell at the appropriate level in the appropriate tab. The level at which this is added is indicated by the name of the special processing section. An example is provided below:

Electrical Special Processing	Special processing for all electrical objects found in sheets, "Devices", "Customer & Service", "Structures", "Annotation" and "Conductors".
Landbase Code Lookups	Special processing for all land base classes found in sheet "Landbase".
Gas Special Processing	Special processing for all gas mapping sheets.

Special Processing Example



Model Build Process for Work Orders

When running the model build process, there can be multiple versions of a given map in the queue of maps to be processed. Map versions must be processed in the order they are submitted to the model build process. If an older version of a map cannot be committed to the model, the system must keep the newer version from being applied.

For single map (version) processing, this is generally not an issue since all the files to be preprocessed are put into one directory; however, when the maps to be processed are provided in model build directories (*i.e.*, work orders), the maps that cannot be committed (*i.e.*, blocked work orders) will become dependencies on any future map or work order that contains any map from the blocked work order.

Configuration of this feature is optional and will be enabled if you define MP_DIRECTORIES and MP_EXTENSIONS. MP_DIRECTORIES will be a list of directories where your model import files are located (i.e., `export MP_DIRECTORIES=$NMS_HOME/data/mp`). MP_EXTENSIONS will be a list of extensions for import files (i.e., `export MP_EXTENSIONS=mp`). If you have multiple extensions or directories, delimit them with either ";"s or spaces.

To configure the preprocessor to support work_order directories, follow the example in OPAL_build_map.ces and make special note of the use of the environment variable *`_wo_dir`*, which is set by the <project>_build_map.ces script to support work order style mutli-partition directory builds. This variable will identify the $OPERATIONS_MODELS/patches sub-directory to write the .mb files to.

PowerFlow Engineering Data Workbook

The PowerFlow Engineering Data Workbook is an Excel spreadsheet used to gather and manage data required by the PowerFlow extensions and other DMS applications that are not generally available within the GIS and Oracle Utilities Network Management System. The Power Flow Engineering Data Workbook maintains data required to run the Power Flow Extensions, Suggested Switching, Optimal Power Flow, Feeder Load Management, and Fault Location Analysis applications.

The Power Flow Engineering Data Workbook defines the required data types, the data tables, and the table schemas. An MS Excel spreadsheet is used for each data type and its corresponding data table. Tabs (worksheets) in the Excel spreadsheet contain a description of the data table and the data table columns. Each data worksheet also contains one or more user-editable tables the user fills for each device type in the data model. The user simply edits the enterable table, adding a new row for each unique device type. For each completed worksheet, the user generates an SQL formatted ASCII text file from a push button on the TOOLS worksheet. The SQL formatted ASCII text files are used to import the Power Flow Engineering data into the Oracle Utilities Network Management System data model.

The Power Engineering workbook maintains and generates the following PowerFlow configuration tabs:

- Sources
- Line Catalog
- Line Limits
- Switch - Fuse Limits
- Power Transformer Impedance
- Power Transformer Taps
- Power Transformer Limits
- Customer Load

- Capacitor Banks
- Customer Hourly Load Profiles
- Distributed Energy Resources

Power Engineering Catalog Data SQLs to be Generated

The Power Flow Engineering Data Workbook will generate a set of customer catalog data SQLs, and those files should be installed in the $CES_DATA_FILES directory (~/sql).

Data file	Description
~/sql/<project>_powerflowengineeringdata.xlsm	This is the latest checked-in version of the Power Flow Engineering Data workbook, to be used for generating the customer catalog data sql files.
~/sql/<project>_pf_sources.sql	Contains data pertaining to equivalent source models for the source nodes in the network.
~/sql/<project>_pf_line_catalog.sql	Impedance details of lines.
~/sql/<project>_pf_line_limits.sql	Line limit details.
~/sql/<project>_pf_switches.sql	Contains nominal ampacity data for switches.
~/sql/<project>_pf_load_data.sql	Contains electrical characteristics of customer loads.
~/sql/<project>_pf_load_profile_feeder.sql	Contains profiles for a full set of feeders. This data is only used if load profiles are configured to use feeder load profiles. THIS IS NOT USED AT THIS TIME.
~/sql/<project>_pf_xfmrtypes.sql	Contains electrical characteristics data for power, step and auto transformers
~/sql/<project>_pf_xfmrtaps.sql	Contains electrical characteristics data for power, step and auto transformers
~/sql/<project>_pf_xfmrlimits.sql	Contains multiple ratings/limits for branch flows based on seasons for transformers
~/sql/<project>_pf_capacitors.sql	Contains electrical characteristics of capacitors
~/sql/<project>_pf_tempswitchcap.sql	Contains electrical characteristics of temperature-regulated capacitors

Data file	Description
~/sql/<project>_pf_hourly_load_profiles.sql	Contains load profiles for load classes (e.g., res, comm, ind). This data is only used if load profiles are configured to use load class profiles.
~/sql/<project>_pf_dist_gen_data.sql	Contains electrical characteristics of distributed generation devices.

Model and Power Engineering Workbook Locations

An example of these workbooks is included in the Oracle Utilities Network Management System Oracle Power and Light example model and configuration included with every release package. You can find these two workbooks in the $CES_HOME/OPAL/Workbooks directory of the Oracle Utilities Network Management System system.

Model Manipulation Applications and Scripts

After a customer has built a model, there may be times when certain scripts or applications may need to be run to clean up errors that have been introduced into the model or to remove obsolete devices or maps. There are several scripts and applications that exist to do this model manipulation. These scripts and applications are described below.

DBCleanup

Most customers should run the DBCleanup application periodically. It examines the modeling database tables and looks for duplicate active rows, orphaned objects, and inconsistencies in the ALTERNATE_VIEWS table. If any of these problems are discovered, the application will attempt to fix the data so that it is consistent with the rest of the database tables.

See also **Troubleshooting Issues with ICP Device Symbology** on page 8-73 for fixing issues with ICP objects.

ces_delete_map.ces

The ces_delete_map.ces script allows the user to remove an obsolete map from the model. It creates a patch that is processed by MBService that will deactivate the map itself and all devices contained in it. This script should be used sparingly.

ces_delete_object.ces

The ces_delete_object.ces script allows the user to deactivate all instances of a single, specified device in all the maps in which it appears. It creates a patch that is processed by MBService to remove all the instances of the device.

ces_delete_branch_obj.ces

The ces_delete_branch_obj.ces script also allows the user to deactivate all instances of a single, specified device from all the maps in which it appears. However, this script directly modifies the modeling database tables, potentially leaving the services in a state that is inconsistent with the current information in the database. After this script is used, either all services should be re-started or MBService should be re-started and then all the maps involved with the deleted object should be re-built. After MBService re-builds the maps, it will send out notifications to the other services to bring them all into sync.

ces_delete_patch.ces

The ces_delete_patch.ces script allows the user to delete a single patch or a range of patches that exist in the database. The script directly modifies the modeling database tables, potentially leaving the services in a state that is inconsistent with the current information in the database. After this script is used, either all services should be re-started or MBService should be re-started and then all the maps involved with the deleted patches should be re-built. After MBService re-builds the maps, it will send out notifications to the other services to bring them all into sync.

mb_purge.ces

The mb_purge.ces script can be used to reduce the size of the modeling tables in the database. It will remove old, inactive rows as specified by the user.

AuditLog

The AuditLog application works with the scripts and applications defined above to keep a persistent record in the database of the data manipulation activities that have been going on when a customer uses any of these scripts or applications. The information is stored in the MODEL_AUDIT_LOG database table and can be useful when trying to help support a customer with corrupted data by helping to provide a better scenario of the activities that might reproduce the problem.

Schematics

Oracle Utilities Network Management System– Schematics can automatically generate orthogonal schematic overviews of the nominal network.

Model Requirements for Schematics

In order to use Oracle Utilities Network Management System Schematics, the following is required of the data model:

- All substations must have the same partition class.
- The substation partition class must only be used by substations.
- All boundaries between feeders and substations are designated with a distinct class of devices.

Schematic Limitations

Since Oracle Utilities Network Management System Schematics uses a splayed-tree representation of the nominal network, it is necessarily geared towards radial networks and will have difficulty representing nominally looped, parallel or meshed areas. Oracle Utilities Network Management System Schematics is also geared towards simple network objects (i.e. a switch) and cannot keep related devices in close proximity (i.e., the internals of a switching cabinet).

Configuring Schematics

- All schematic configuration is controlled via command-line options which are passed to the schematic-generator, schematica. The script that contains the configuration is normally called <project_name>_create_schematics.ces
- The script must perform these three actions:
- Remove any previous schematic import files.
- Call schematica with all of the configured options.
- Process all generated import files.

The following table describes all of the command line options.

Command Line Option	Description
-addStop <list of classes>	Include these classes as well as those specified via -stop
-balanceSubstations	Shift feeders around a substation until there are similar NUMBERS on each valid side.
-boundingBoxCls <class name>	Create a box of this class to indicate the substation-overviews extents. If unset, the box is not drawn.

Command Line Option	Description
-boundingBoxLabelCls <class name>	Create a label of this class, with the substation's name. Default is branch.
-branchWidth <dist>	Distance between two network branches that share a common upstream port. (see Figure 1 below)
-camelHumpHeight	Relative height (in terms of tier-height) of conductor-crossover bumps. Value between 1 and 0. (See Figure 4 below)
-camelHumpWidth	Relative width (in terms of branchWidth) of conductor-crossover bumps. Value between 1 and 0.
-classesToLabel <list of classes>	List of classes for which the schematic-generator should create and place annotation.
-connectionClass <class name>	Device class to use when creating a branch to span two or more non-conductor devices. (Must be a non-electrical branch)
-coordSystem <#>	The coordinate system the schematic generator will assign all schematics. Should not be an existing value. Defaults to the current maximum coord_system + 1
-db <DBService prefix>	Force the schematic-generator to use the DBService that has the specified prefix (*i.e.*, -db MB will use MBDBService)
-dch	(Disable Camel-humps) Don't create camel-humps where conductors cross
-defaultConductorSymbology <valid symbol class>	Use this symbol class when attempting to write out any conductor that has a symbol class of 0.
-defaultFeederDirection <north|south|east|west>	If the schematic-generator is unable to determine the direction for a feeder, it will use this value. No default. If this option is NOT specified, the schematic-generator will ignore any feeder for which it can not determine a direction.
-deviceGaps <class name> <scale factor>	Scales all diagrams of the specified classes by the specified amounts
-deviceHeight <#>	Size of all non-conductor electrical branches. (See figures following this table.)
-deviceScaling <class name> <scale> <offset>	Scale all diagrams of the specified classes as well as shift them along their parent feeder's axis. Default scaling is 1.0, default offset is 0.0
-excluded <class name>	Any classes specified here will be excluded from the generated schematic map.
-fastCrossovers	Use a faster, but less accurate algorithm to determine where conductors intersect.

Command Line Option	Description
-fbdBounded	Use this option if all feeder-heads have FID on one port and an FBD on the other.
-feederDirection <north|south|east|west>	Have ALL feeders extend in the specified direction.
-feederHeight <#>	Minimum distance between a substation and the first device of a feeder. (See figures following this table.)
-feederNameTable <table name> <column name>	The specified table for each feeder's FID, annotated with the value found in the specified column. For single-circuit schematics, the feeder name is used as part of the generated map's name.
-feederOffset <#>	Distance between adjacent feeders. Default is branchWidth*10. (See figures following this table.)
-feederPrefix <comma-delimited list of strings>	Only process substations whose map names contain the specified strings
-feederTextScale <#>	Amount to scale all feeder-name annotation.
-geographicSubstations | -gs <table name> <column name>	Use this option when all substations are modeled in the geographic world. Schematica will search the specified table for all classes listed in -substationNodeClasses and set the substation name based on the value in the specified column. (This option must be used in conjunction with -substationNodeClasses)
-globalScaleFactor <#>	Increases the size of all objects and all overviews by this amount.
-ignoreUnconnectedOpenPoints	Treat any open switch that has no connections on either port as if it were a regular switch when determining what devices to prune from the feeder schematic.
-intersubOffset <#>	Minimum distance between parallel sub-to-sub conductors. Defaults to tier-height or device-height*2, whichever is greater. (See figures following this table.)
-invisibleClasses <list of classes>	List of classes (that never have symbology, *i.e.*, zero-impedence conductors) that the schematic-generator should ignore when attempting to connect a feeder to its parent substation.
-labelClass <class name>	Use this text class when creating device annotation if the class <device_class_name>_t2 does not exist. Text class to use for all generated annotation. (See figures following this table.)
-mapPrefix <prefix>	Prepend all generated schematic maps with this prefix. **Required**.

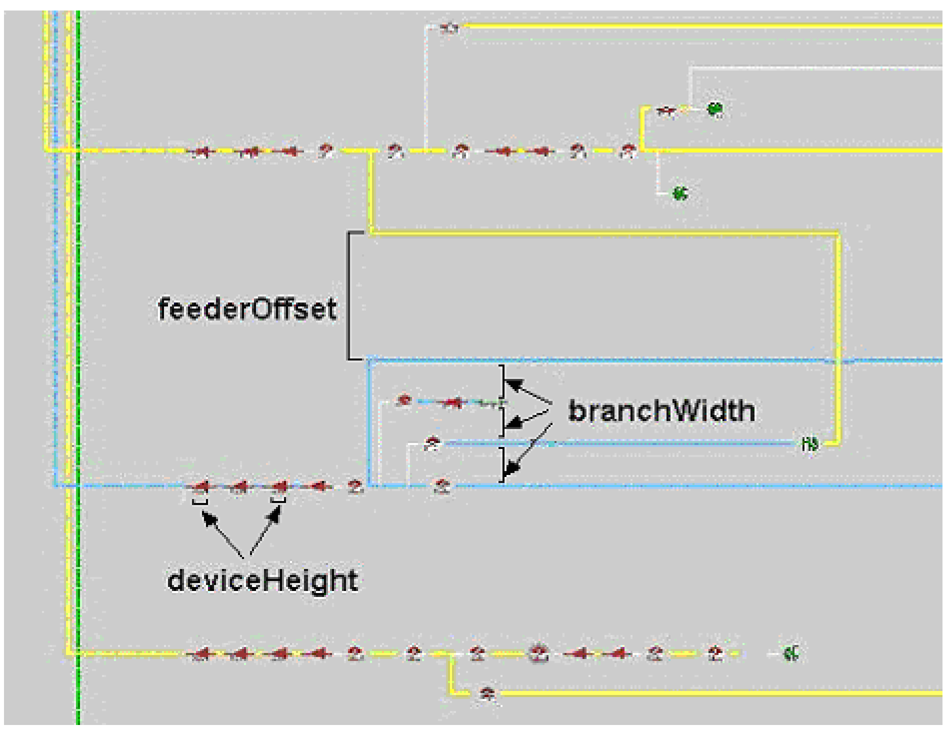
Command Line Option	Description
-maps <list of map names>	The list of schematic map filenames (excluding file extension) that should be rebuilt, with spaces between the map names. Only these maps will be rebuilt.
-noFeederToFeeder	Do not connect up feeder-to-feeder tie-points. (See figures following this table.)
-noIntraFeederConnections	Do not connect up bypass tie-points
-noPrune	Keep all devices in a feeder, not just those attached to open-points.
-noSubstations	Do not draw substations. Instead draw all feeders in the same map in one or more rows. This option is only used when drawing all feeders in one map. To draw each feeder in a separate map, do not use this option; use the -substationName with an argument to group by feeder_name. (See -maxRowWidth)
-noSubToSub	Do not connect up sub-to-sub tie-points.
-orientation <ANY|HORIZONTAL|VERTICAL|ROUND_ROBIN|NONE>	Align all feeders according to the value. (ANY = normal feeder directions, HORIZONTAL = move all north/south-ward feeders to east/west, VERTICAL = move all east/west-ward feeders to north/south sides, ROUND_ROBIN = evenly distribute the feeders around the substation, NONE = move ALL feeders to side specified by "-feederDirection") Default is ANY.
-overviewName <string>	The names of all resulting schematic maps will take the form <map prefix>_<overview name>_<substation name>
-placeSubsByConnection	Attempt to position substations with the greatest number of common connections closest to each other.
-priorityClasses <list of classes>	Keep the specified list of classes as close to the main trunk of the generated schematic tree as possible.
-ptncls <#>	Partition class to use for all generated schematics. **Required**.
-reorientDeviceClasses <list of classes>	Ensure that diagrams for the specified classes are always oriented from left to right. (Use this if symbols appear upside-down.)
-scaleFactor <#>	Multiplies the size of all conditions by this amount.
-skipEmptyFeeders	Do not draw feeders that contain an exceedingly small number of devices (< 10 devices)

Command Line Option	Description
-sort <GEO|SPAN>	Arrange feeders either geographically (using only the anchor points of each feeder) or arrange them to minimize the distance feeder-to-feeder tiepoint connections must span. Values: GEO|SPAN GEO = geographic ordering, SPAN = minimal spanning tie points. Default is GEO
-startAtFID	Use when all feeder heads are modeled to have an FID attached.
-stop <list of classes>	List of all device classes the schematic-generator should not trace past.
-subSpacing <#>	Minimum distance between substations. No default. (See figures following this table.)
-substationBoxCls <class number>	Create a box of this class-type around the substation. No default. If not specified, there will be no visible box around the substation.(See figures following this table.)
-substationBoxSize <#>	Create a square of the specified size and scale the original substation schematic to place inside. Default is 1000. (See figures following this table.)
-substationName <database table name> <table column name>	Do not model substations. Instead, search the specified table for each feeder's FID and group them into substations based on the values in the specified column. For example, if the column is the feeder_name column, each feeder will have its own schematic map. It could also be the substation_name, which would group all feeders form one substation together. Or it could be any other column in the specified table or view, as desired.
-substationNodeClasses | -snc <list of class names>	When used in conjunction with -geographicSubstations, it specifies what type of nodes to initiate substation tracing from. Generally, the value should be SRC.
-substationPtnCls <#>	Only process substations with this specific partition-class. No default.
-substationTextScale <value>	Amount to scale the size of the substation label.
-substationTransitionClass <list of classes>	The set of classes that designate the transition between feeder and substation. Defaults to hyper_node.
-tapDeviceOffset <value>	Distance to offset single devices from the main trunk.
-textOffset <#>	Distance (along the feeder's main axis) to pull all device annotation. Default is 0. (See figures following this table.)

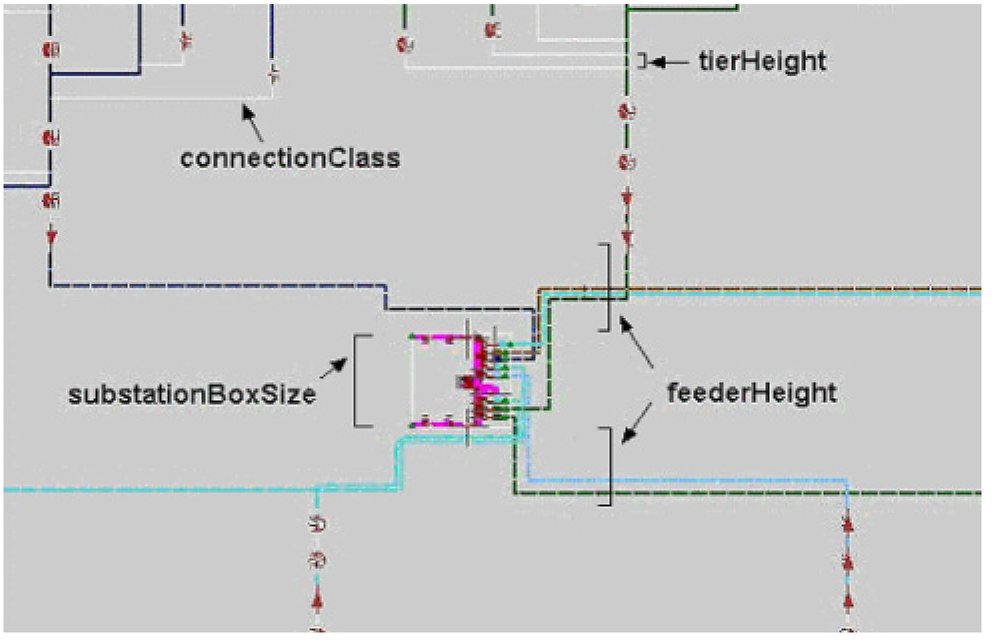
Command Line Option	Description
-textScale <#>	Scale all device annotation by this amount.
-textScaleSubstationDevices <value>	Amount to scale the text size for substation device annotations. Default value is 1.0. Alternative command is **-tssd**.
-tierHeight <#>	The distance (along the feeder's axis) a conductor will span. Default is 50. (See figures following this table.)
-tilebasedmaps | -tbm	Calculate the geographic orientation of each feeder based on the coordinates of the all open points, not on the base map's extents.
-validFeederStartClass <list of classes>	List of classes that designate the start of a feeder. **Required**.
-voltage <minimum voltage> <max voltage>	Only process devices that fall into the specified voltage range.
-weightClass [<class name> <weight>…>	Tells the schematic-generator to process certain classes of objects sooner when creating its internal schematic tree. If weight < 0, process later. If weight > 0, process sooner.

Note: <list of classes> format: [-]class name[+],[-]class name[+],....] [-] exclude this class (and all descendents if '+' is used) [+] include all descendents.

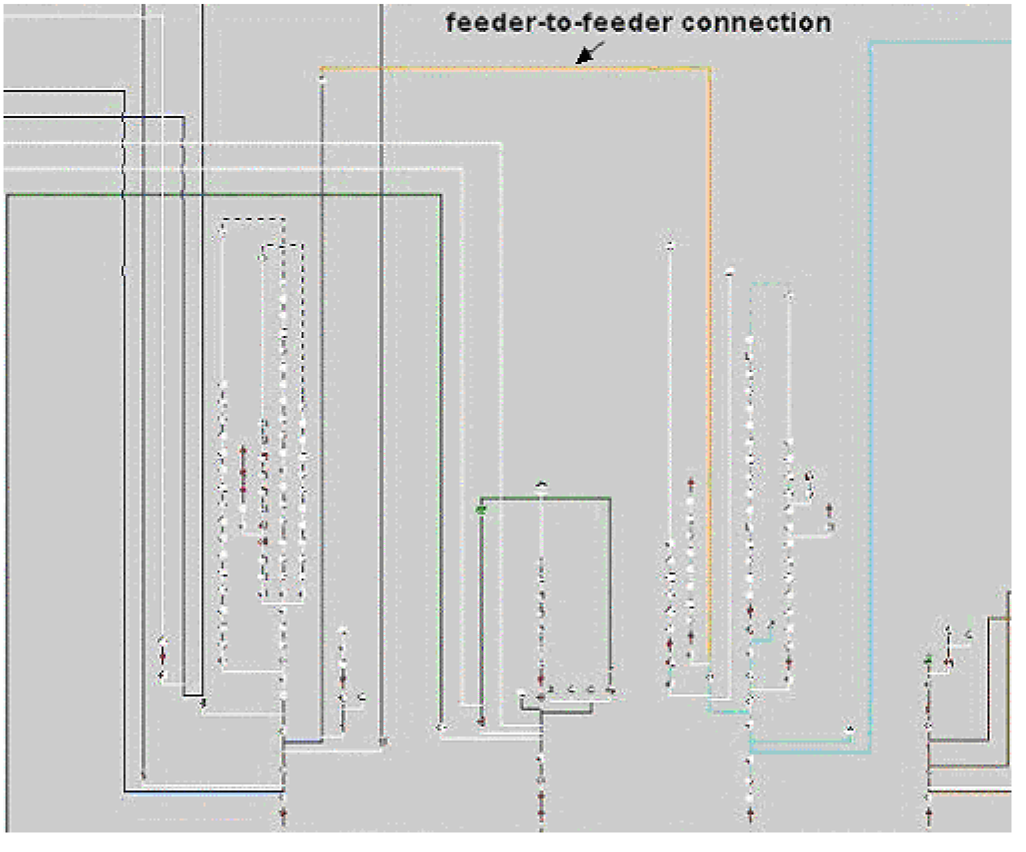
The following figure shows the deviceHeight, branchWidth, and feederOffset.



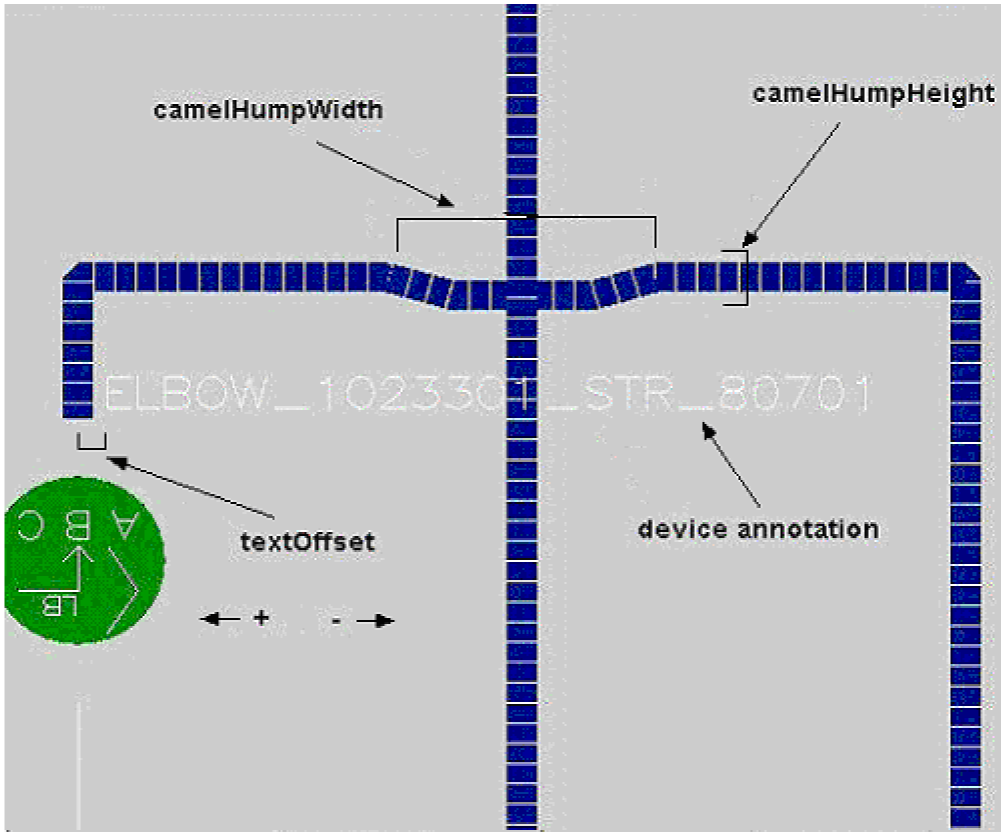
The following figure shows substationBoxSize, feederHeight, tierHeight, and connectionClass.



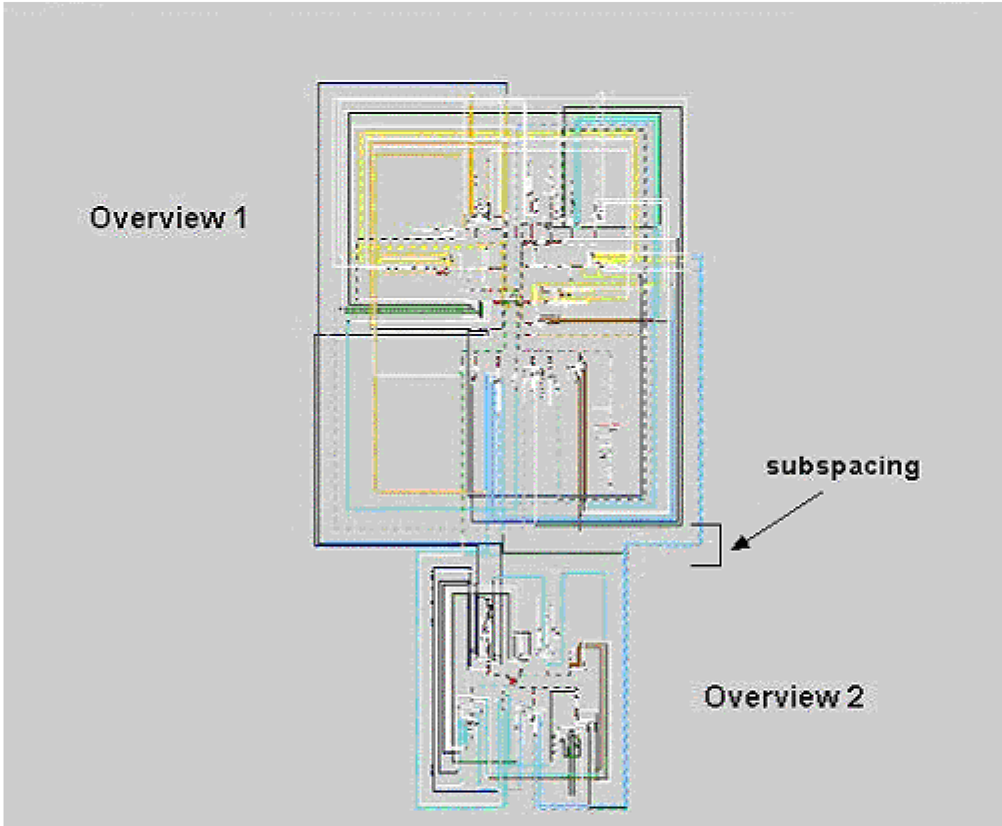
The following figure shows a feeder-to-feeder connection.



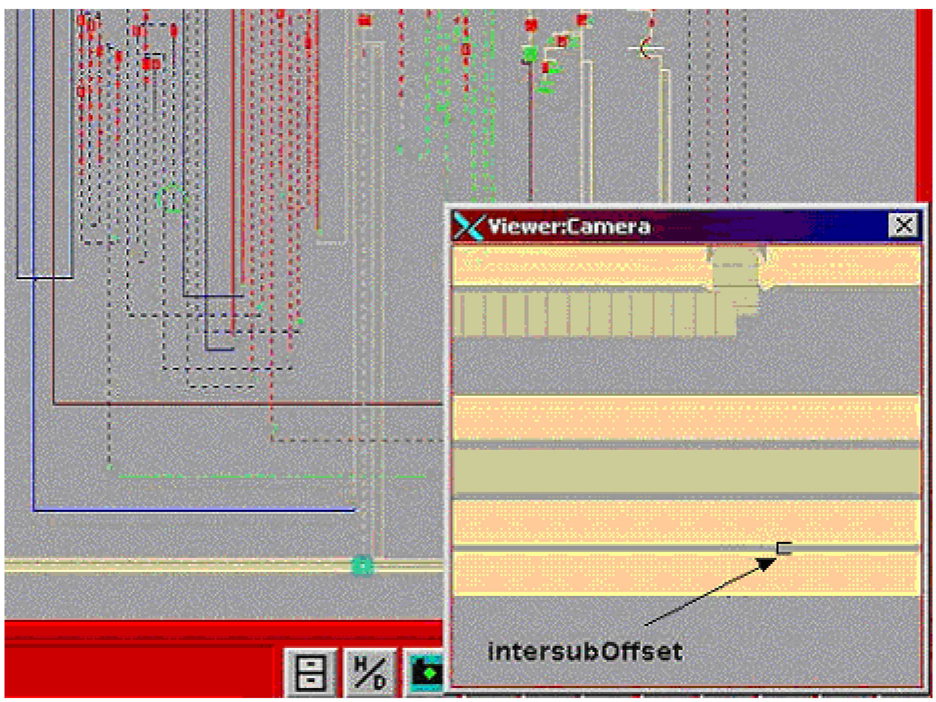
The following figure shows camelHumpWidth, camelHumpHeight, device annotation, and textOffset.



The following figure shows subspacing.



The following figure shows intersubOffset.



Generating Schematics

To create schematics, the customer-specific script <project>_post_build.ces must have a call to <project>_create_schematics.ces.

The Post-Build Process

After the build process has processed the final map, it calls <project>_post_build.ces. If there is an entry for create-schematics, it calls it at this time.

Creating the Import Files

Once invoked, the schematic generator loads in the entire nominal network model and attempts to group all feeders with their parent substations. After it finishes determining the layout and spacing for all feeders and substations, it writes out one import file for each substation.

Processing the Import Files

After the schematic-generator creates the import files, the schematic script compares the most recent previous version of each file. If no changes are detected, it skips the map. Otherwise it proceeds to build the import file as per the normal model-build process.

Aggregate Devices

Aggregate devices are devices that are linked together in the model so that the user only needs to select one device and display one Control Tool to operate any number of associated devices.

Model Requirements for Aggregate Devices

Use the Distribution Model workbook to populate the aggregate_devices table. This can contain multiple records for a single controller device, so long as the seq_num is unique for each. In this manner, you can connect a single device to aggregate backfeed devices or replicate a more complex construct with a set of switches that all operate from a single Control Tool.

Field	Format	Comments
controller_cls	NUMBER(6)	The handle class of the controller id
controller_idx	NUMBER(12)	The handle index of the controller id
sec_cls	NUMBER(6)	The handle class of the secondary device
sec_idx	NUMBER(12)	The handle index of the secondary device
seq_num	NUMBER(6)	The sequence of the secondary device.

This is an attribute table, so the standard h_cls, h_idx, partition, birth, birth_patch, death. death_patch, and active columns are also required.

In Construction Pending / Device Decommissioning (ICP)

Oracle Utilities Network Management System supports the modeling and visualization of devices that are in-construction as well as devices that are marked for decommissioning. ICP can be used for commissioning new construction (such as road widening) and should not be used for nominal-state changes (such as feeder load balancing).

Device Lifecycles

In a GIS system, a device will fall in to one of four possible states:

Device State in GIS	Description
Install	Objects that are proposed construction or new objects to be commissioned at a future date
Existing	All objects that are in the GIS as-built and commissioned
Remove	Objects that are commissioned today and are part of the active model however there is a construction plan to remove these objects
Retired	All objects that have been completely de-commissioned. These devices will not exist in the real-time system.

Model Requirements for ICP

In order to use In Construction Pending (ICP), each affected device must have an additional value listed in their physical_properties entry inside the import file, as shown below:

Device State in GIS	Required Physical_Properties Value in Import files
Install	Construction
Existing	NA
Remove	Decommission
Retired	NA

The model preprocessor calculates these values and writes them out into import files.

> **Note:** Model Extractors must be modified to not filter out devices in the "Install" state.

Model Builds and Commissioned/Decommissioned Devices

The Commissioning Tool moves devices between "Not Commissioned" and "Commissioned" as well as "Not Decommissioned" and "Decommissioned".

If an operator commissions a device, marked as Construction, a model build will not reset the commissioning state (i.e., Subsequent model builds will not undo changes made by the Commissioning Tool).

Effect of ICP Devices on Network Topology

Devices affect the network's topology as follows:

Device State	Commissioned / Decommissioned	Does Device affect Network Model
Install	Not commissioned	No.
Install	Commissioned	Yes. As normal existing device.
Remove	Not decommissioned	Yes. As normal existing device.
Remove	Decommissioned	No.

ICP Device Symbology

The Viewer will hide certain ICP-marked devices and display certain ICP devices with additional symbology.

Device State	Commissioned / Decommissioned	Default Visibility	Special symbology
Install	Not commissioned	Hidden	Yes.
Install	Commissioned	Visible	Yes
Existing	NA	Visible	No
Remove	Not decommissioned	Visible	Yes
Remove	Decommissioned	Hidden	Yes

Note: See the User Guide section on "Using the Operator's Workspace Viewer" for more information on ICP symbology and how to use the Commissioning Tool.

Troubleshooting Issues with ICP Device Symbology

If you notice that some pending construction and pending decommission objects are missing conditions and are not hiding correctly with the Hide/Display option in the Web Workspace Viewer, you can run DBCleanup with the -fixICP option.

To see the objects that are missing conditions:

```
DBCleanup -fixICP -showMe
```

To add the conditions:

```
DBCleanup -fixICP
```

DDService is required to be running. No services need to be stopped or restarted when using this option.

You can run **`DBCleanup -fixICP -skipMBSCheck`** if you are only performing this ICP cleanup routine and no other model-related routines.

Note: running **`DBCleanup -fixAll`** does **NOT** run the **`-fixICP`** option.

Auto Throw-Over Switch Configuration (ATO)

Oracle Utilities Network Management System supports the modeling and visualization of Auto Throw-Over (ATO) devices. Critical customers such as hospitals, manufacturing, financial and emergency services, require higher level of power quality and reliability. These customers are normally provided with a primary and backup source of power to improve the reliability. Utilities deploy automatic throw over devices to switch the load to backup source when the primary source is not available. Often these devices have automatic restoration feature where the load is fed by the primary source when primary source is energized after an outage.

Model Requirements for ATOs

In order configure ATOs in the Oracle Utilities Network Management System, the Model Build process needs to know what two devices are controlled by the ATO controller. One device must be identified as the primary or preferred feed, which would be normally closed, and the other device would be the secondary or alternate feed, which would be normally open. These relationships and control behaviors are modeled in the ATO_CONTROLLERS table, as shown below:

Field	Format	Comments
H_CLS	N	Class part of the ATO controller handle. Required.
H_IDX	N	Index part of the ATO controller handle. Required.
PARTITION	N	ATO controller partition.
CONTROL_FUNCTION	V32	ATO control function identifier. Required. Values: • 2dev – 2 ATO Devices and no auto-restore • 2dev_arc – 2 ATO Devices, auto-restore, no momentary on restore operation • 2dev_momentary_acr – 2 ATO Devices, auto-restore, and will create a momentary on restore operation
ATO1_CLS	N	Class part of the handle of the primary ATO device. Required.
ATO1_IDX	N	Index part of the handle of the primary ATO device. Required.
ATO2_CLS	N	Class part of the handle of the secondary ATO device. Optional.
ATO2_IDX	N	Index part of the handle of the secondary ATO device. Optional.
PARAM1	N	Delay (in seconds) until primary ATO device is opened during throwover - Optional.
PARAM2	N	Delay (in seconds) until secondary ATO device is opened during auto-return (ignored by control function “2dev” but column presence is still required) - Optional.
PARAM3	N	Delay (in seconds) between operating primary and secondary ATO devices. If not configured, there is no delay. -Optional.

Field	Format	Comments
BIRTH	D	Birth date of when the object is activated into the model
BIRTH_PATCH	N	Patch which activated this object
DEATH	D	Death date of when the object is de-activated from the model
DEATH_PATCH	N	Patch which de-activated this object
ACTIVE	V1	Active flag

Summary Object Configuration

Summary Objects are objects in one world (i.e., Geographic World) that reflect events or conditions in another world (i.e., Substation World). For example, a substation fence in the geographic world may display the conditions existing on objects within the substation in the internal world view of the substation (i.e., an outage on a breaker in the substation would be reflected on the fence in the geographic world).

To configure this functionality, you need to configure three areas of the model:

1. Verify that summaryobjects is on the DDService in the `~/etc/system.dat file`.
2. Verify that product_srs_rules.sql has a config rule for summaryObject set to "yes".
3. Verify that all object classes you wish to have summary events reflected on are in the project condition rules file (i.e., substation_fences).
4. Substation fences, when build, must define a location in the .mb file. For example:

```
ADD substation_fence 2 {
    LOCATION = <10210.2>;
    ALIAS[OPS] = "SUB_Lake";
    DIAGRAM[1022] (1022) = {
        SYMBOLOGY = 101;
        HEIGHT = 500.000000;
        GEOMETRY = {
            (2270311.397232,460321.122269),
            (2270311.397232,459286.466476),
            (2271217.293103,459286.466476),
            (2271217.293103,460321.122269),
            (2270311.397232,460321.122269)
        };
    };
    ATTRIBUTE[Latitude]="  40.92498";
    ATTRIBUTE[Longitude]=" -81.40776";
};

ADD LOCATION <10210.2> {
    NAME = "SUB_Lake";
    DESC = "Lake Substation";
    REFERENCE = (2270311.397232,460321.122269);
};
```

5. All objects in the substation partition that you want the events and conditions reflected on the substation fence must belong to the same location. For example:

```
ADD rack_circuit_br 1500 {
    PHYSICAL_PROPERTY = SUB;
    VOLTAGE = 13800;
    NCG = 63;
    PHASES = 7;
    LOCATION = <10210.2>;
```

```
PORT_A = <444.2523.2>;
PORT_B = <444.2522.2>;
ALIAS[GIS] = "Circuit Breaker.270";
ALIAS[OPS] = "BR241XFM";
DIAGRAM[1094] (1094) = {
    RANK = 65544;
    SYMBOLOGY = 10507;
    HEIGHT = 500.000000;
    GEOMETRY = {
        (205.811207,412.902928),
        (205.811207,391.655951)
    };
};
ATTRIBUTE[gmd_location] = "Lake Substation";
ATTRIBUTE[gmd_comment] = "0.0000";
```

Symbology

The Viewer displays all model objects and conditions as symbols, either vector symbols or raster symbols. This symbology system is made up of four types of symbols (with the indicated symbol identifier (SIN) range):

- Firm Symbols (30,000 - 99,999)
- Non-Firm Symbols (100 - 2100)
- Soft Symbols (2100 - 29,999)
- Pixmap Symbols (2100 - 29,999; same as Soft Symbols)

Firm and Non-Firm symbols are generally used for linear objects like conductors, roads, and boundaries. Soft symbols and pixmap symbols are generally used for devices (switches, transformers, capacitors, etc.) and other “point” devices.

Firm Symbols

Firm symbols have a five digit SIN based on the pattern: **LSDCC.** Each digit defines an aspect of the 1D symbol that is drawn in the Viewer.

- **L**: long dash length
- **S**: space length
- **D**: dash pattern
- **CC**: color code

Firm symbols are indicated by SINs ranging from 30000 to 99999.

L - Long Dash Length. The long dash length is the continuous part of the line between the spaces and short dashes, if any. This digit determines how many pixels the long dash will be. It must be 3 or greater to classify as a firm symbol.

D Value	Description	Sketch
0	No short dashes	____________________
1	One point, one pixel	________ . _________
2	Two points, one pixel each	_______ . . ________

D Value	Description	Sketch
3	Three points, one pixel each	_____ . . . ______
4	One short dash, 1 * S	_______ _ ________
5	Two short dashes, each 1 * S	______ _ _ _______
6	Three short dashes, each 1 * S	_____ _ _ _ ______
7	One short dash, 2 * S	_______ __ _______
8	Two short dashes, each 2 * S	______ __ __ _____
9	Three short dashes, each 2 * S	____ __ __ __ ____

S - Space Length. The space length is the gap between long dashes and short dashes. This digit defines the pixel length of the space as L=S*2. An S value of zero results in a solid line even when the dash pattern is greater than zero.

S Value	Length (pixels)
0	0 (No Space)
1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18

D - Dash Pattern. The short dash pattern defines the number and size of short dashes in the line. There can be from zero to three short dashes in each line pattern. The short dashes can be one pixel points, space sized dashes or double space sized dashes.

CC - Color Code. The line color is specified by a two digit color code.

CODE	COLOR	CODE	COLOR	CODE	COLOR	CODE	COLOR
100	black	21	grey60	43	coral2	65	darkgreen
01	white	22	grey70	44	yellow1	66	seagreen
02	red	23	grey80	45	yellow2	67	firebrick
03	yellow	24	grey90	46	blue4	68	tomato
04	green	25	red1	48	orange1	69	lightgoldenrod
05	cyan	26	red2	49	orange2	70	goldenrod1
06	blue	27	red3	50	brown4	71	hotpink1
07	magenta	28	red4	51	magenta1	73	magenta4
08	orange	29	limegreen	52	magenta3	74	chocolate4
09	pink	30	turquoise	53	steelblue1	75	wheat1
10	tan	31	violet	54	steelblue2	76	thistle4
11	grey	32	violetred	55	cyan4	77	steelblue
11	gray	33	deeppink	56	orange4	78	maroon4
12	navy	34	aquamarine	57	yellow4	79	coral1
13	brown	35	khaki	58	moccasin	80	deeppink1
14	purple	36	goldenrod	59	ltpink	81	laurellee
15	salmon	37	gold	60	deepskyblue	82	slategrey
16	grey10	38	coral	61	mediumaqua	83	royalblue
17	grey20	39	maroon	62	snow1	84	orchid
18	grey30	40	wheat	63	blue1	85	dkorange
19	grey40	41	green3	64	cadetblue	99	eaudenil
20	grey50	42	green4				

Non-Firm Symbols

Non-firm symbols have a four digit SIN based on the pattern: **LLCC.**

- **LL**: line style
- **CC**: color code

Non-firm symbols are indicated by SINs ranging from 100 to 2100; if the SIN is less than 1000, assume a zero before the first digit.

LL - Line style. Choose a line style number based on the desired dash pattern and background color. Dash pattern refers to the alternating number of pixels to draw of specified color and background color. The first number draws the prescribed color, CC; the second number draws the background color; the third number, if any, draws the prescribed color and so on.

Line Style Number	Dash Pattern (pixels)	Background Color
1	None	Transparent
11	10,1	Transparent
12	10,1,2,1	Transparent
13	10,1,2,1,2,1	Transparent
14	10,1,2,1,2,1,2,1	Transparent
15	20,10	Grey30
16	50,10,10,10	Grey30
17	75,10,10,10,10,10,10,10	Grey30
18	2,4	Transparent
19	15,15	Black
20	15,15	White

CC - Color Code. The color codes are the same as those listed firm symbols. Use the color code to prescribe the foreground color of the dash pattern. The SIN 106 is drawn in the Viewer as a solid blue line. The SIN 1614 is drawn in the Viewer as a dashed line with 50 pixels of purple, 10 pixels of gray30, 10 pixels of purple and 10 pixels of gray30.

1D Width Multiplier

The width multiplier increases the thickness of the firm or non-firm 1D symbol. Add one or more digits ranging from 1 to 29999 to the base SIN to increase the width of the line drawn on the Viewer. The multiplier increases the width of the line proportionally to the map scale so that the line width increases and decreases with zoom level. If no multiplier is specified, the line width is always one pixel regardless of zoom level. The actual width of the symbol in pixels is calculated at run time. Note that the results of the multiplier vary with each model.

The width multiplier is added to the beginning or left side of the base SIN starting with the sixth digit. Since non-firm SINs only have four digits, a zero must be added prior to adding the multiplier. For example, 5001324 is a non-firm symbol with base SIN 1324. The width multiplier is 50. The extra zero is a placeholder only. The firm symbol 5045733 with base SIN 45733 also has a width multiplier of 50. Divide the symbol id number by 100,000 to determine the width multiplier.

Soft Symbol Definitions

Soft symbols are classified as a point or line. They are indicated by SINs ranging from 2101 to 29999. Symbol definitions, in the <project>_SYMBOLS.sym file, have a regular pattern consisting of a header and a body. The first line of the header is called the *header line* and is followed by additional required key attribute lines.

Symbol Header

Header Line

The first header line always begins with **`SH`** followed by symbol type, code, and name in the following format:

```
SH <symbol_type> <symbol_code> <symbol_name>
```

- **symbol_type**: a point (P) or a line (L)
- **symbol_code**: the unique symbol identification number (SIN)
- **symbol_name**: a text string that names the symbol.

Examples

- Point transformer with SIN 2200:

```
SH P 2200 xfmr
```

- Line switch with SIN 2201:

```
SH L 2201 switch
```

A1 <x> <y>

A required record that defines the first anchor point of a line symbol or the only anchor point for a point symbol.

Examples

- Add the default focus point for the transformer:

```
SH P 2200 xfmr
A1 0 0
```

- Add the first anchor point for the switch:

```
SH L 2201 switch
A1 -10 0
```

A2 <x> <y>

A required record for line symbols that defines the second anchor point.

The anchor points determine the default focus point for line and point symbols. The default focus point for line symbols is the midpoint between the two anchor points, A1 and A2. The anchor point, A1, is the default focus point for point symbols. Once the drawing coordinates are determined, the symbol is scaled and rotated around its focus point.

Example

- Add the second anchor point for the switch with default focus point (0,0):

```
SH L 2201 switch
A1 -10 0
A2 10 0
```

CF <foreground_color_number> <background_color_number>

A required record that defines the colors used for filled objects and double dash lines. The foreground color for filled objects is the line color and the background color is the fill color.

Examples

- Set the transformer foreground color to 3 (yellow) and background color to 0 (black):

```
SH P 2200 xfmr
A1 0 0
CF 3 0
```

- Set the switch foreground color to 1 (white) and background color to 0 (black):

```
SH L 2201 switch
A1 -10 0
A2 10 0
CF 1 0
```

Symbol Body

SB

The line containing only **SB** denotes the end of the symbol header and the beginning of the symbol body containing description lines. Each description line defines a new aspect of the soft symbol including color changes, line style changes, draw actions, and movements.

> **Note:** the end of the symbol body is designated by the end of the file or the beginning of a new symbol.

Examples

- Transformer:

```
SH P 2200 xfmr
A1 0 0
CF 3 0
SB
```

- Switch:

```
SH L 2201 switch
A1 -10 0
A2 10 0
CF 1 0
SB
```

The following sections describe valid actions for the symbol body.

PEN

s <color_number> - Sets the pen to a specified color that cannot be overridden by the Viewer selection color. If a symbol drawn with this pen is selected in the Viewer, it does not blink or change colors.

Example

- Set the switch pen color to black:

```
SH L 2201 switch
A1 -10 0
A2 10 0
CF 1 0
SB
s 100
```

SO <color_number> - Sets the pen to a specified color that can be overridden by the Viewer selection color. If a symbol drawn with this pen is selected in the Viewer, it blinks or changes color.

Example

- Set the transformer pen color to black:

```
SH P 2200 xfmr
A1 0 0
CF 3 0
SB
SO 100
```

Line

W <width> - Specifies the line width. The results of this value varies for each model.

Example

- Set the switch's line width to 1:

```
SH L 2201 switch
A1 -10 0
A2 10 0
CF 1 0
SB
s 100
W 1
```

L <line_style_number><length><length><length>… - Sets the line style and dash pattern.

Valid values for `<line_style_number>` are:

- 1 - solid
- 2 - dash; alternate between specified color and transparent
- 3 - double dash; alternate between foreground color and background color

The `<length>` parameters are optional. They specify the segment lengths for the dash pattern. There can be many `<length>` parameters, but the last one must be equal to zero.

Example

- Set the switch's line style to 1 (solid):

```
SH L 2201 switch
A1 -10 0
A2 10 0
CF 1 0
SB
s 100
W1
L 1
```

D <x1><y1><x2><y2> - Draws a line symbol between two points (x1, y1) and (x2, y2).

Example

- Draw a solid line with a width of 1 starting at (0,0) ending at (6,0) for the switch:

```
s 100
W1
L1
D 0.0 0.0 6.0 0.0
```

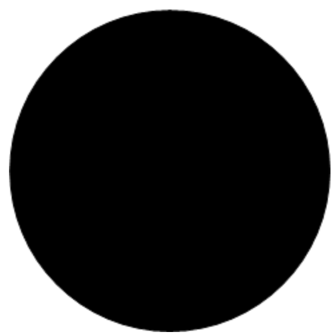
CIRCLE

C <x><y><radius> - Draws a filled circle with center (x, y) and a specified radius.

Examples

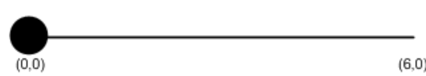
- Draw a black filled circle at (0,0) with radius 2.5:

```
C 0 0 2.5
```



- Draw a black filled circle at (0,0) with radius .3:

```
s 100
W1
L1
D 0.0 0.0 6.0 0.0
C 0.0 0.0 .3
```

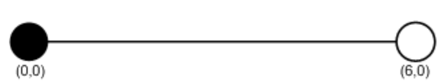


c <x><y><radius> - Draws an open circle with center (x, y) and a specified radius.

Example

- Draw an open circle with center (6,0) and radius .3:

```
s 100
W1
L1
D 0.0 0.0 6.0 0.0
C 0.0 0.0 .3
c 6.0 0.0 .3
```



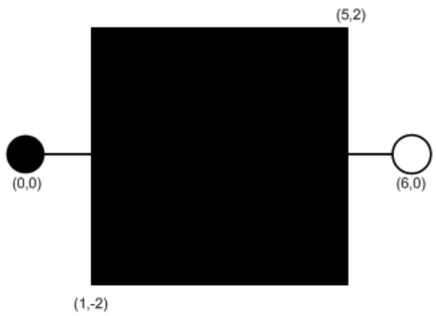
BOX

B <x1><y1><x2><y2><angle> - Draws a filled box between the opposite corners, (x1, y1) and (x2, y2), with the specified angle of rotation.

Example

- Draw a black filled box

```
s 100
W1
L1
D 0.0 0.0 6.0 0.0
C 0.0 0.0 .3
c 6.0 0.0 .3
B 1.0 -2.0 5.0 2.0 0.0
```



b <x1><y1><x2><y2><angle> - Draws an open box between the opposite corners, (x1, y1) and (x2, y2), with the specified angle of rotation.

Text

t <height><width><vertical_justification><horizontal_justification> - Sets the height and width of the text at a specified justification. Vertical and horizontal justification have the following values:

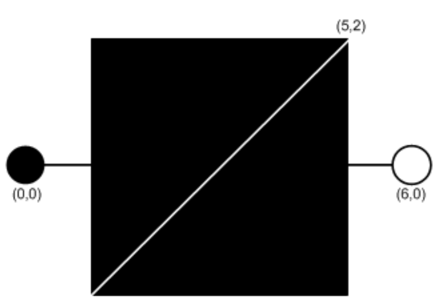
- 0 - left or bottom
- 1 - center
- 2 - right or top

The text is drawn with the 'T' record, but the 't' record must be defined first.

Example

Draw a diagonal line with a white pen color; define text attributes with vertical justification = 0 and horizontal justification = 0

```
s 1
D 1.0 -2.0 5.0 2.0
t 1.0 1.0 0 0
```

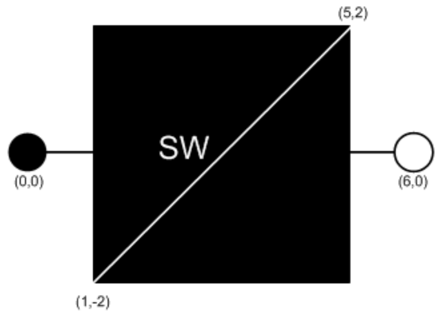


T <x><y><angle>"<string>" - Draws the text, "<string>", at (x, y). The text formatting is defined by the 't' record and must be defined prior to the 'T' record.

Example

- Draw the text "SW" at (2,0) with specified text attributes.

```
T 2.0 0.0 0.0 "SW"
```



Polygon

M <x><y> - Defines the first coordinate for a filled polygon. This record must precede the 'P' action.

Example

- Set the pen color to grey70 and define the first point of the polygon for the transformer:

```
s 22
M 0.0 2.0
```

P <x><y> - Defines the next coordinate for a filled polygon. Use this action to specify as many points as necessary. This record follows the ‘M’ action and precedes the ‘F’ action.

Example

- Set the remaining points of the polygon for the transformer:

```
P -1.7 -1.0
P 1.7 -1.0
```

F <x><y> - Defines the last coordinate for a filled polygon. This record follows the ‘P’ action and is the same as the ‘M’ action. It finishes and fills the polygon.

Example

- Finish and fill the polygon. The result is the transformer symbol, xfmr.

```
F 0.0 2.0
```

ARC

a <x><y><radius><begin angle><end angle> - Draws a circular arc at (x, y) with radius from begin angle to end angle.

SCALED OBJECTS (line, circle, box, polygon)

SW <w> - Defines the scaled line width as a percentage of the distance between anchor points.

N - No scale option for lines, circles, boxes or polygons. This must be defined on the same line as the object this record applies to.

Z <A1> or <A2> or <x><y> - Overrides the default focus point of a line, circle, box or polygon. This must be defined on the same line as the scaled object this record applies to.

Hover Text

H "<string>" - Adds a tooltip that is activated when the user's mouse hovers over the symbol.

Example

- Add "Probable Service Outage" to the probable service outage (PSO) condition symbol.

```
SH P 4001 probable-service-outage
C#
A1 0 0
CF 1 0
SB
s 4
C 0 0 70
SO 1
B -47.36 47.36 47.36 -47.36 45
s 100
t 80 40 1 1
T 0 0 0 "PSO"
H "Probable Service Outage"
```

Pixmap Symbols

Use $NMS_CONFIG/jconfig/ops/viewer/properties/RasterSymbols.properties to specify the image file to use for a given symbol. For example:

```
# This contains a mapping of symbols that should be
# displayed as raster images
# The first file is the normal image.  If a second image is listed,
# it is for the selected image. example:
#14042=sym_green_truck.gif,sym_green_truck_sel.gif

#14042=sym_crew.gif
#14043=sym_red_green_truck.gif
#14044=sym_orange_truck.gif
```

SVG Symbols

SVG symbols offer more complicated device, condition, and outage symbols than the standard `.sym` symbols, and can be edited and enhanced more easily using standard tools.

To use SVG symbols, add your `.svg` symbols to your `$NMS_CONFIG/jconfig/ops/viewer/images` directory and add references to them to the `$NMS_CONFIG/jconfig/ops/viewer/properties/SVGSymbols.properties` file.

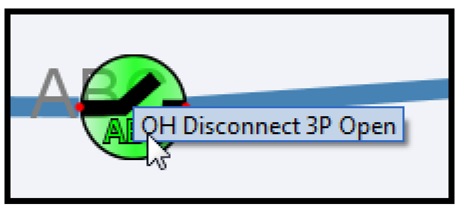
You can optionally use the `ces_svg_populate_properties.ces` script to automatically add records to the `SVGSymbols.properties` file.

The `SVGSymbols.properties` file maps the `SYMBOLOGY_STATE.symbology_id`, `CONDITION_RULES.symbol_num`, `ANALOG_RULES.symbol_num`, or `QUALITY_RULES.symbol_num` database symbol numbers to the `.svg` symbol file. For example:

```
4002=Probable_Device_Outage.svg
4004=Real_Device_Outage.svg
14802404=closed_C_fuse.svg
12602700=open_switch.svg
```

The SVG `<title>` element is used for hover text.

```
<svg>
...
<title>OH Disconnect 3P Open</title>
```



Converting SYM Files to SVG

The ces_sym_to_svg.ces script converts .sym files to .svg files. The script will create one .svg file for each .sym symbology preserving the symbology name, ID, anchor points, and all graphic components.

Usage: `ces_sym_to_svg.ces` *`<input.sym file>`*

To convert your sym files, perform the following steps:

1. Run `ces_sym_to_svg.ces.` The script will output the .svg files in the directory where it is run.
2. Move the svg files to the $NMS_PROJECT/jconfig/ops/viewer/images directory
3. Run ces_svg_populate_properties.ces to populate the SVGSymbols.properties file.

Updating Symbology

After the symbology file has been updated the following command will update the java application server:

```
Action any.publisher* ejb reload_symbology
```

Any running clients will need to be restarted to pick up the new symbology.

Symbology Mapping

The SYMBOLOGY_STATE Table

The SYMBOLOGY_STATE database table contains the mapping from each device's symbology class, state, and off-nominal status to the symbol used for it.

Column	Description
context	At this time, always 'RT_TOPOLOGY'
symb_state_class	The symbology class, as defined in the DIAGRAM_OBJECTS table
state	The bitwise switch status, from 0 to 15, or the conductor energization status, from 0 to 23. The conductor energization statuses are as follows: 0: (Unknown) 1: (Unused) 2: (Ignored) A pending construction or decommissioned conductor 3: (Energized) A conductor that is energized 4: (De-Energized) A de-energized condcutor 5: (Unused) 6: (Parallel) A conductor fed by a set of parallel feeders 7: (Didirectional) A conductor fed from both sides, either due to a loop or the direct path of a parallel 8: (Meshed) A conductor fed by at least one feeder marked with an MID 9: (Degraded) A partially energized conductor 10: (Phase A) Phase A is hot. Used in Phase Coloring mode only 11: (Phase B) Phase B is hot. Used in Phase Coloring mode only 12: (Phase C) Phase C is hot. Used in Phase Coloring mode only 13: (Phase AB) Phases AB are hot. Used in Phase Coloring mode only 14: (Phase AC) Phases AC are hot. Used in Phase Coloring mode only 15: (Phase BC) Phases BC are hot. Used in Phase Coloring mode only 16: (Multistate) A combination of Grounded, Faulted, Parallel, etc. on separate phases 17: (Grounded) The conductor is grounded 18: (Faulted) The conductor is grounded and energized, or fed by two different phases connected together 19: (Trace) The conductor is part of the trace results requested by the user 20: (Isolated) The conductor is in an isolated segment 21: (Delegated) The conductor is in a delegated zone 22: (Secure) The conductor is in an secured segment 23: (Abnormal Radial) The conductor is fed by only one MID feeder
off_nominal_flag	0/1: Whether the specified state/symbology_id is for an off-nominal position
symbology_id	The symbol ID to use. This is either the conductor coloring identifier or the symbol ID found in the symbol file

The QUALITY_RULES Table

The QUALITY_RULES database table contains the mapping from device class and quality code to the desired symbol or text.

Column	Description
priority	The priority of the symbol. If more than one symbol matches exactly, the lowest priority rule will be used
value	The integer value of the quality
string	The string to display, usually a single character
description	A description of the quality
color	The color of the text to use
location	The location (1-9) to use
symbol	The symbol ID to use, or 0, if text is to be used
off_nominal	Unused

The CONDITION_RULES Table

The CONDITION_RULES database table contains the mapping from device class and condition type to the desired symbol.

Column	Description
condition	The condition class name, or 'digital' for digital SCADA measurements
class	The device class
devsymbol_num	The digital attribute number, used only for digitals. 0 otherwise
priority	The priority of the symbol. If more than one symbol matches exactly, the lowest priority rule will be used
location	The location (1-9) to use. These correspond to upper-left, upper, upper-right, left, center, right, lower-left, lower, lower-right
status	The condition status
symbol_num	The symbol to use

The ANALOG_RULES Table

The ANALOG_RULES database table contains the mapping from device class and SCADA analog id to the deisred text position.

Column	Description
attrib_num	The attribute number
priority	The priority of the symbol. If more than one symbol matches exactly, the lowest priority rule will be used
location	The location (1-9) to use

Power Flow Data Requirements and Maintenance

This section describes the Power Flow Extensions Data Input process and the customer data requirements and associated database schema. The data will be used by all DMS applications, including Power Flow Extensions, Suggested Switching, Optimal Power Flow, Feeder Load Management, Short Circuit Analysis, and Fault Location Analysis. It is intended to assist the customer during the Power Flow Extensions data modeling process. It is meant as an introduction, rather than a comprehensive reference. It includes the following subsections:

- Power Flow Extensions Data Import Process – this section describes the basic process and data sources used during the data import process.
- Modeling Device Data – this section explains the basic data requirements and relevant database tables required to model device data.
- Modeling Load Data – this section explains the basic data requirements and relevant database tables required to model load data.
- Catalog tables, Required GIS attributes, Power Flow Attribute Views, Default Data Tables, and Configuration Tables – these sections contain the table schemas required for the Power Flow Extensions.
- Power Flow Engineering Data Workbook – this section describes the power flow engineering workbook contents, and workbook maintenance and update process
- Power Flow Engineering Data Maintenance – this section provides a list of high level command line options that can be used with PFService to dynamically update the engineering data

Power Flow Extensions Data Import Process

Data for power flow applications is built as part of the normal model-build process.

The catalog tables contain electrical characteristic data for represented device types in the electrical distribution system. Customer data for these catalog tables are captured through the Power Flow Engineering Data workbook.

The device attribute views are created based on data in the GIS attribute tables and the customer-provided catalog tables. These views map each relevant network device to class and index attribute keys in the *network_components* table. They may have to be tailored as per the project and the availability of data in both the GIS and catalog tables. The data requirements for each device type are discussed in the *Modeling Device Data* section below.

Modeling Device Data

The creation of device-specific attribute data-sets (either database views or tables) is an important step in the Power Flow Data Modeling process. As mentioned earlier, this process relies on the availability of data from two sources, viz., GIS attribute tables and device catalog tables. This section discusses the device information that is used for this process.

Sources

The customer must provide an equivalent source model for each source node defined in the data model. These source nodes represent constant voltage buses that are used to determine energization of the system and are generally located at feeder heads, the substation secondary bus generation sources, or the substation primary side bus. The required equivalent source parameters must represent an equivalent impedance looking up into the transmission system from the source node in question, in addition to voltage magnitude and angle. It is possible to model equivalent sources as zero impedance (i.e., 'infinite bus'), but this will impact the accuracy of short circuit calculations provided by the Power Flow Extensions.

The *pf_source_view* is created based on the data available in the source GIS attribute table (*att_generator*), the device id tables (*network_nodes, feeders*), and the customer catalog table (*pf_equivalent_sources*).

Line Impedences

The customer must provide line data in one of three ways:

- The first and preferred way is to provide the phase impedance data for each three-phase line type. The phase impedance data must be the self and mutual impedance and shunt susceptance for each phase. The phase impedance data must be provided in the *pf_line_ph_impedance* table. Oracle Utilities Network Management System Power Flow Extensions supports the modeling of symmetric and asymmetric lines. Lines are considered symmetric when the 3 phases have the same conductor type. Lines are considered asymmetric when the 3 phases have at least two different conductor types.
- The second way is to provide the sequence impedance data for each line. The sequence impedance table data must be the positive and zero sequence impedances and shunt susceptance for each line. The sequence impedance data must be provided in the *pf_line_seq_impedance* table.
- The third way only applies to overhead lines and is used to provide the conductor and construction types for each line. The preprocessor uses Carson's Modified Equations to calculate phase impedance data from these inputs. This data must be provided in the *pf_cond_types*, *pf_oh_cond_types*, *pf_oh_conductor_spacing*, and *pf_oh_cond_catalog* tables. *pf_cond_types* contains the conductor characteristics. *pf_oh_cond_types* contains the phase-wise conductor type description (as catalogued in the *pf_cond_types* table). The *pf_oh_conductor_spacing* table contains the phase-wise spacing (coordinates) information for an overhead line type. The *pf_oh_cond_catalog* table contains the conductor types (as per *pf_oh_cond_types*) and conductor spacing type (as per *pf_oh_ conductor_spacing*) mappings of all overhead lines.

The GIS attribute table for overhead lines is *att_oh_elec_line_seg*.

Transformers and Regulators

Oracle Utilities Network Management System Power Flow Extensions supports explicit modeling of multiple forms of power transformers, such as auto transformers, load-tap-changing transformers, step-up/step-down transformers and regulators. Each of these types of transformers and regulators require transformer characteristic data provided in the *pf_xfmr_types* and *pf_ltc_xfmr* tables, which are read by the Model Preprocessor which in turn writes the PF_XFMR and PF_XFMR_TANKS tables, which are read by the Power Flow Service.

Capacitors and Reactors

Shunt (capacitor/reactor) parameters must be provided from the customer's data source(s), typically the GIS. These parameters are defined in the PF_CAPACITOR_DATA table, which will be accessed by the Model Preprocessor, which in turn will write the PF_CAPACITORS table, which will be read directly by the Power Flow Service.

Oracle supports the following types of shunt regulation:

- **Voltage switched capacitors**: local or remote bus regulation
- **Current switched capacitors**: local line or cable regulation
- **KVAR switched capacitors**: local line or cable regulation
- **Power-factor switched capacitors**: local line or cable regulation
- **Temperature switched capacitors**: on and off temperature
- **Time of day switched capacitors**: on and off time of day.
- **Fixed capacitor:** no regulation
- **Capacitor sequence control**

The supplied data for each shunt must indicate which type of regulation is to be used and the corresponding control attributes.

Modeling Loads

Load modeling consists of basic load data which is used to determine average loading levels and load profile data, which is used to provide detailed information for load variation over time.

Basic Load Data

During modeling efforts, loads must be assigned to specific equipment types. The preferred approach is to insert a load (supply node) at the secondary of each underground and overhead distribution transformer. Supply nodes may also be created at primary metering points for cases where there is no transformation or transformation is unknown or customer-owned.

For each load, a utilization factor can be specified, which represents the average loading level for the rated size of the transformer. For the most accurate power flow results, this data should be based on per-instance consumption data, which can often be obtained from historical billing information by dividing the total energy consumption of attached customers by the billing period and transformer rating.

The power flow data for load is defined using the table pf_load_data, which is read by the Model Preprocessor, which in turn writes the pf_loads table, which is read by the Power Flow Service.

Load Profile Data

Load profile data is used to model how load changes over time. A single load profile represents the change in load levels over a 24-hour period. Multiple profiles may be associated with a single load to represent different load behavior for different types of day (e.g., weekday, weekend) and for different seasons. The use of load profile data improves the accuracy of the DMS applications by providing more realistic loading scenarios for the current or predicted analysis time period. For example, profiles are used to verify switch plans, determine suggested switching recommendations, and generate daily and seasonal peak limit alarms.

The Oracle Utilities Network Management System supports a variety of sources of load profile data such as load class profiles or individual transformer profiles. Once processed, all profile data is placed in the *pf_load_interval_data* and *pf_loadtype_data* tables. The profile_id field of the *pf_loads* table points to the data in these tables.

Load Class Profiles

Load class profiles represent typical load changes over time for a particular type or class of load, such as residential, commercial and industrial. This type of profile data can be obtained from general sources, or the customer can collect this data from typical customers or feeders. When using load class profiles, the load level at each load point is determined by combining the rated kVA with the load utilization factor and the class profile associated with that load. Load class profiles are useful where detailed data for each load is unavailable.

Transformer Profiles

Modern Meter Data Management (MDM) systems make it possible to collect detailed power usage histories for each customer. By aggregating individual meter loads to each service transformer, it is possible to create detailed load profiles for each transformer location. This data can be derived from either representative historical conditions or using predictive values, if the MDM system has this capability.

When using transformer profiles, all load data is derived from these profiles and basic load data such as the utilization factor is not used. The load profile input data can include both kW and kVAr values for load over the 24 hour period.

Catalog Tables

The catalog tables identified in this section must all be populated by the customer. The Power Flow Data Engineering Excel workbook should be used as a template to assist the customer in identifying source data locations (planning power flow data, database tables etc.), defining a data export mechanism, and specifying the Oracle table names, columns, and data formats into which the source data must be imported. See the example workbook in the Oracle Utilities Network Management System product directory location: $CES_HOME/OPAL/workbooks.

Configuration Tables

pf_seasons

This table will store the seasonal peak load information and define the seasons. One entry in this table is required for every season of every load zone. A load zone consists of a group of all loads that have their load profiles maintained according to the same temperature measurement point.

There could be only one load zone for the entire system, or there could be several. The customer directly populates this table from seasonal data.

Attributes			Description
season_Number	Varchar2	20	Season number
Zone	Varchar2	20	Zone number
Season_peak	Varchar2	20	Season peak load in KVA
peak_day	Varchar2	20	Day of seasonal peak load
peak_month	Varchar2	20	Month of seasonal peak load
peak_load_period	Varchar2	20	Load period of seasonal peak
peak_day_type	Varchar2	20	Day type of seasonal peak
peak_temp	Varchar2	20	Peak temperature in °F or °C

srs_rules

Attributes			Description
Nom_Temp	Varchar2	50	Nominal temperature for use when maintaining the load profile.
DAYTYPE_X	Varchar2	100	Identifies the day type of the current day, where 'X' is the number of the day type represented by this field.

DAYTYPE_X Attribute

The following table provides examples for the DAYTYPE_X attributes of the pfs_rules table.

Parameter	Value
DAYTYPE_0	WEEKDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
DAYTYPE_1	WEEKEND, SUNDAY, SATURDAY
DAYTYPE_2	SEASONALPEAK

Day types can be configured by day name and days. Each day should appear exactly once in the set of daytypes. In the table above:

- All weekdays are of day DAYTYPE_0.
- All weekend are of DAYTYPE_1.
- Seasonal peaks are of DAYTYPE_2.

For more information on srs_rules table configuration, refer to the **Chapter 14**, **Distribution Management Application Configuration**.

Power Flow Service High Level Messages

Several high level messages are available for data maintenance and troubleshooting.

- To refresh the DMS applications with the updated data, you must restart PFService:

```
Action any.PFService restart
```

- To send configuration parameters to the log (e.g., inheritance, command line options, PFS rule set, and configuration rule set), execute:

```
Action any.PFService dump
```

- To send the topology.dat file of an island to the log, execute:

```
Action any.PFService dump_model <handle>
```

- To re-initialize system source data with PFService running, execute:

```
Action any.PFService updatesystemsources
```

- To re-initialize reload system load data with PFService running, execute:

```
Action any.PFService updatesystemloads
```

- To re-initialize system capacitor data with PFService running, execute:

```
Action any.PFService updatesystemshunts
```

- To re-initialize system transformer data with PFService running, execute:

```
Action any.PFService updatesystemxfmrs
```

- To re-forecast Feeder Load Management data with new equipment ratings after a model build, execute:

```
Action any.PFService flm reforecast
```

- To re-enable islands after a model build to correct non-converged islands, execute:

```
Action any.PFService reenable_island <source alias or handle>
```

Spatially Enabling the Data Model for Advanced Spatial Analytics

The ces_parameters table contains a set of attributes that are used to enable the NMS electrical model for Oracle Utilities Advanced Spatial Analytics. The following table describes these attributes.

ces_parameters attribute	Description
MBS_GEO_SRID	The Oracle Spatial reference ID for the geographic spatial layer.
MBS_GEO_MINX MBS_GEO_MAXX MBS_GEO_MINY MBS_GEO_MAXY	The minimum and maximum values for the two coordinate systems.
MBS_LL_SRID	The Oracle Spatial reference ID for the lat/long spatial layer.
MBS_GEO_CSMAP_COORDSYS	The CS_MAP-defined geographic coordinate system.
MBS_LL_CSMAP_COORDSYS	The CS_MAP-defined lat/long coordinate system.
MBS_LL_MINX	Min X value of data, used for spatial indexing
MBS_LL_MINY	Min Y value of data, used for spatial indexing
MBS_LL_MAXX	Max X value of data, used for spatial indexing
MBS_LL_MAXY	Max Y value of data, used for spatial indexing
MBS_LL_TOL	Tolerance value, used for spatial indexing
MBS_LL_XTYPE	X Coordinate type (Typically X or LATITUDE)
MBS_LL_YTYPE	Y Coordinate type (Typically Y or LONGITUDE)
MBS_GEO_TOL	Tolerance value, used for spatial indexing
MBS_GEO_XTYPE	X Coordinate type (Typically X or LATITUDE)
MBS_GEO_YTYPE	Y Coordinate type (Typically Y or LONGITUDE)

NMS CIM Import and Export Tools

NMS has two tools to support import and exporting CIM data: cim2mp and CIMExporter, respectively.

CIM Import

The CIM import processor, *cim2mp.ces*, feeds directly into the standard NMS model build process. It takes a CIM-formatted file and converts it into an NMS model preprocessor (mp) file. Once the files are in the .mp file format, the Model Engineer configures the rest of the model interface just as they would with any GIS-supplied .mp file.

Usage:

`cim2mp.ces` *`cim_file mpfile`*

Example:

```
cim2mp.ces 3513.rdf 3513.mp
```

CIM Export

The CIM export tool, *CIMExporter.ces*, exports a specific set of components from the NMS .mb file in CIM/IEC .xml/.rdf file format. The resulting file should be able to be imported by a CIM-compliant model consumer.

Usage:

`CIMExporter.ces` *`mbfile cim_file`*

Example:

```
CIMExporter.ces 3513.mb 3513.rdf
```

The CIMExporter.ces configuration file, CIMExport.properties, is located in the $NMS_HOME/ sql directory. The product version of this properties file is installed by default. Copy the product file to your project/sql directory and make any changes needed. Run **nms-install-config** to install the project/sql version into the $NMS_HOME/sql directory.

Note: running nms-install-config will overwrite the product version.

CIMExport.properties can be used to:

1. map NMS classes to CIM classes
2. specify the NMS attributes to map to CIM attributes
3. enable catalog lookup for powerflow line values

Preparing the NMS Model for Oracle Utilities Customer Self Service

The Oracle Utilities Customer Self Service application reads NMS materialized views to display NMS data. These materialized views are created in the <*project*>_CSS_setup.ces script and refreshed using the <*project*>_CSS_refresh.ces script. Note that the user_sdo_geom_metadata table needs to be updated for the new materialized views with the source table rows' *diminfo* and *srid* values.

Materialized Views

GEOGRAPHIC_OUTAGES

The GEOGRAPHIC_OUTAGES materialized view is created from the JOBS, DIAGRAM_OBJECTS, and the NETWORK_COMPONENTS tables.

Column Name	Data Type	Description
outage_type	VARCHAR	A description of the outage type. "Probable Service Outage", as mapped from the JOBS.status
num_customers_out	NUMBER	The number of customers out, as mapped from the JOBS.user_cust_out.
begin_time	DATE	The outage begin time, from JOBS.begin_time.
est_rest_time	DATE	The outage estimated restore time, from JOBS.est_rest_time.
last_update_time	DATE	The outage last updated time, from JOBS.last_update_time.
cause	VARCHAR	The outage cause, if available.
geometry	MDSYS.SDO_GEOMETRY	The geographic geometry, from DIAGRAM_OBJECTS.geo_geometry.

GEOGRAPHIC_OUTAGE_AREAS

The GEOGRAPHIC_OUTAGE_AREAS materialized view is created from the zip_codes, jobs, supply_node_log, and customer_sum tables.

Column Name	Data Type	Description
area	VARCHAR	The zip_code of the outage area, from the CUSTOMER_SUM.zip_code.
cust_served	NUMBER	The number of customers in the area, as summed from CUSTOMER_SUM.customer_count for that area.
cust_out	NUMBER	The number of customers out in the area, as summed from CUSTOMER_SUM.customer_count for the supply_nodes with outage,
earliest_begin_time	DATE	The earliest outage begin time in the area, from the JOBS.begin_time.
latest_est_rest_time	DATE	The last JOBS.est_rest_time for the area.
last_update_time	DATE	The last JOBS.last_update_time for the area.
geometry	MDSYS.SDO_GEOMETRY	The ZIP_CODES.geometry for the area.

GEOGRAPHIC_OUTAGE_STATUS

The materialized view GEOGRAPHIC_OUTAGE_STATUS is created from the JOBS table.

Column Name	Data Type	Description
report_date	DATE	The date this view was created – SYSDATE.
cust_served	NUMBER	The sum of all CUSTOMER_SUM.customer_count records.
cust_out	NUMBER	The sum of all active JOBS.user_cust_out records.
num_outages	NUMBER	The number of distinct active JOBS table records.

Model Build File Export to XML

NMS has two utilities for exporting model data files to XML. The resulting .xml files is defined by the mb.xsd XML schema file, which is found in `$CES_HOME/product/sql/`.

MB Export to XML

To export an existing .mb file to XML, use the following command:

Usage:

```
mb2xml [-debug][-partitionClasses class_names ...]
        -classDirectory class_directory_paths
        -xml xml-file -mb mb-file
```

Options	Arguments	Description
-debug		enable debugging
-partitionClasses	*class_names*	additional partition class name(s)
-classDirectory	*class_directory_paths*	directories to find class file(s)
-xml	*xml-file*	output xml file name
-mb	*mb-file*	input mb file name

Note: Recall that the resulting .xml file is defined by the XML schema file: `$CES_HOME/product/sql/mb.xsd`.

Schematic Data Export to XML

The schematic data file export takes a .mad file and exports it to XML.

Usage:

```
maa2xml [-debug] -classDirectory class_directory_paths
        -xml xml-file -maa maa-file
```

Options	Arguments	Description
-debug		enable debugging
-classDirectory	*class_directory_paths*	directories to find class file(s)
-xml	*xml-file*	output xml file name
-maa	*maa-file*	input maa file name

Note: Recall that the resulting .xml file is defined by the XML schema file: `$CES_HOME/product/sql/mb.xsd`.

Chapter 9

Database Maintenance

As general maintenance, you should establish a schedule to analyze tables, defragment your database, and purge historical/unnecessary data (then re-analyze the tables). You should also set up a schedule to backup your database and archive the backups.

This chapter describes all of these processes as well as the process of reconciling differences in database requirements when you upgrade your model to a new release of Oracle Utilities Network Management System.

It includes the following topics:

- **Oracle Configuration**
- **Purging Historical Data**
- **Applying Product Migrations**
- **Applying Custom Migrations for Oracle Utilities Network Management System Integrators**

Oracle Configuration

The following database settings are suggested for at least a minimum level of performance for an Oracle database. Any of these suggestions can be disregarded if an experienced Oracle DBA determines that other settings may offer better overall system performance. However, if any changes are made to any suggested parameters, performance of the system may be affected.

Indexes

Indexes should not be placed on the same physical disk as the data resides. If disk striping is being used then this requirement is not as critical, and may be ignored if enough disks are being employed.

Generating Statistics

As mentioned in a previous section of this chapter, tables should be analyzed periodically. The frequency can be determined by an experienced DBA, but it is suggested that this be done when the model or outage data changes substantially. This ensures that the Oracle statistics will be kept up to date for all of the database tables.

Most tables can be analyzed any time, but some tables are cleared and repopulated while services are running. These tables should not be done as part of a batch process; instead, they should be analyzed only while fully populated.

You should analyze these tables only after a full set of Feeder Load Management forecasts:

- flm_cap_bank_details
- flm_dev_violations_warnings
- flm_dist_gen_details
- flm_equiv_source_details
- flm_fdr_load
- flm_fdr_load_details
- flm_fdr_tie_points
- flm_feeders
- flm_islands
- flm_model_errors
- flm_reg_transfmr_details
- flm_solutions.

Oracle Parameter settings

The Oracle Utilities Network Management System requires the Oracle RDBMS has enough memory to support the expected end user performance. Oracle RDBMS can automatically manage shared memory, but it is suggested that the following parameter be set to define the total amount of memory that is available.

- sga_target – This parameter should be set to at least 1G and could be set higher depending on the size of the database

Make Tablespaces Locally Managed

Dictionary managed tablespaces are more expensive on performance. It is suggested that the Oracle Utilities Network Management System tablespaces be setup as locally managed.

Block Size

If possible, the disk block size of the database should be a minimum of16K, but could be set larger on recommendations from an experienced DBA.

Purging Historical Data

As tables continue to grow, many of their rows become “inactive.” The “inactive” data could be historical outage data (completed and/or cancelled outages) or old model build data that is no longer needed.

You should develop a plan to purge the extraneous data from the operational tablespaces (back it up or delete it) on a regular basis. After the data is purged, re-analyze these tables. This process requires proper planning and design because you do not want to lose important information required for reporting or troubleshooting.

Guidelines and Considerations

When developing your plan, it is helpful to understand how the purging process works. This script can purge obsolete data from model build tables or update Oracle statistics for the tables. The age of data is determined by the DEATH column.

From the facilities tables in the operations database, the usage statement is:

```
mb_purge.ces [-rows <integer>]
    [-days <integer> | -date <MMDDYY> ]
    [-purge] [-analyze]
    [-table <table_name>]
    [-debug]
    [-mdlmsg]
```

Option	Description
-purge	Sets the flag to purge the tables otherwise purging will not occur.
-table <table_name>	Specifies single table to purge.
-rows	Number of rows to be deleted at one time, minimize this number to avoid filling up the rollback segments. (Optional) default value: 10,000
-days	Number of days to retain for deathed patches. it may be desired to keep the last 7 days of deathed patches.
-date MMDDYY	Remove all deathed rows before this date
-debug	Sets the debug mode toggle, all debug output will be placed in CES_LOG_DIR/mb_purge_<date>.log (Optional)
-mdlmsg	Incrementally purge rows from the ces_model_msg table. Note this incremental delete operation could take a long time if the ces_model_msg table has grown without bound over a period of time. If you really don't use or care about what is in the ces_model_msg table (generally model build related info, errors, and warnings) you could well save substantial time by just truncating the ces_model_msg table directly. (Optional)
-analyze	Analyze the tables or table specified. If purging, the analyze process will occur after all purging is completed.

Compatibility

An Oracle Utilities Network Management System schema is not backward compatible with Oracle Utilities Network Management System applications. Schema changes occur and are modified as the code and database move forward in time.

For example, it is unlikely that a database which has been migrated or built at version 1.7.10 code level will work with version 1.8.0 code level. However, data models are forward compatible, because Oracle Utilities Network Management System applications can migrate the database forward, making the necessary changes.

Thus, when backing up the database, you should note the Oracle Utilities Network Management System release level that was last operating against the database dump. That way, if there are other systems with older code, the data model is not imported into those systems and problems are not introduced.

Software

The Oracle Utilities Network Management System software is likely to be the most static data on the system. It should only be changing with upgrades. The need for software backup is generally low if the software is installed on several machines locally, but a weekly backup may be needed if there are maintenance scripts and SQL files being updated.

Map Files

Map files are replicated on a number of machines throughout the network, but they will change frequently. Data model files should be backed up once per week at minimum or nightly for frequently changing files.

Applying Product Migrations

The Apply Product Migrations process migrates the model of an older Oracle Utilities Network Management System release to that of a new software version. Based on a release level identifier, the migration process determines the differences between the current model and that of a new release. After the installation of a new release of software, and the loading of a copy of your existing production database, you will need to do the following:

- Execute the `$CES_HOME/bin/ces_setup.ces` script.

This script will call another script called `ces_apply_migrations.ces`, which determines the differences between the release level of the software and the model database. This script then determines the required and optional migrations by accounting for differences in the release database requirements.

Manual Product Migrations

If a manual migration is required, the `ces_setup.ces` script will stop at that point and alert the user of the required manual migration. When this occurs, please see the corresponding manual migration file in the `$NMS_HOME/migration/manual` directory for details on what is required for this migration. The files in this directory are named `<####>.txt`, where <####> is the bug or problem report (PR) number.

The `$NMS_CONFIG/migration/data/<project>_config_ready.dat` file serves as a "sign-off" document for the Oracle Utilities Network Management System project team. As you determine that a manual migration has been completed (or is not needed for your system), you must add the corresponding bug numbers to the `$NMS_CONFIG/migration/data/<project>_config_ready.dat` file by entering one bug number per line. Once you have edited this file, you can run `$CES_HOME/bin/nms-install-config` to copy it to the `$NMS_HOME/migration/data` directory or manually copy the file there if you prefer. This signals the migration script that this particular manual migration has been completed. Once the file has been properly copied to `$NMS_HOME/migration/data`, you need to rerun the `ces_setup.ces` script. Continue this process until all manual and automated migrations are executed.

Note: The bug numbers indicated in the manual migration may not be listed in the Product Fix Document (PFD) supplied with the release. The migrations always refer to an original bug, which is associated with a particular release; any other releases that receive the fix will have a separate bug number (*i.e.*, a "copy-bug"). When resolving manual migration issues, always refer back to the text files placed in the $NMS_HOME/migration/manual directory and not the PFD document associated to that bug fix.

Command Line Options

The ces_apply_migrations.ces script can be initiated directly from the command line in order to view some of the things that it will be doing when started from the ces_setup.ces script. The following table describes all of the command line options for this script.

Option	Description
`-debug`	Displays debug information.
`-showme`	List all processes that would be executed, but do not actually execute any programs or SQL files.
`-needConfig`	Displays a list of migrations that are required by a project.
`-listMigrations`	Displays a list of migrations needed without applying them.

Note: The `ces_apply_migrations` script should not be run without any command-line arguments since that would cause the migrations to actually be executed. The command-line arguments listed above are to be used with the script so that it can be run in a "show only" mode but won't actually do the migrations.

Installing Product Migration Files

The data files that are required for the migration process are installed in the `$NMS_HOME/migration/data` directory. After making changes to the project-specific `$NMS_CONFIG/migration/data/<project>_config_ready.dat` file and an optional special `$NMS_CONFIG/migration/data/<project>_migration.dat` file, run `nms-install-config` script to install them into the `$NMS_HOME/migration/data` directory.

The Product Migration Process

The `ces_apply_migrations.ces` script determines the database differences by comparing the database release level in the `CES_PARAMETERS` table with the software release levels found in the software_release_id.dat and software_release_levels.dat files. Based on these differences, it will create a list containing all of the necessary migrations.

The migration process, or `ces_apply_migrations.ces`, finds the necessary migrations in the `$NMS_HOME/migration/data/pr_migration.dat` file and the `$NMS_HOME/migration/data/product_pr_migration.dat` file, which contains the list of PRs, releases, patch levels, and configuration types. If there are project-specific migrations, then a optional `<project>_pr_migration.dat` file is also used.

The pr_migration.dat files resemble the following example:

PR	Release	Patch	Required	Config Required	Script Exists	ConfigType
----	-------	-----	--------	------	------	----------
19254	5.5	3	Y	Y	Y	config_sql
19831	6.0	3	Y	N	Y	schema_sql

The following table describes the pr_migration.dat file columns.

Column	Description
PR	Bug or Problem Report (PR) number for the migration.
Release	Migration release level, two numbers not including the first digit. For example, release 1.8.1 would be just 8.1 in this field.
Patch	Migration patch level. If the release if 1.8.1.2, then the Patch would be 2.
Required	Whether or not this migration is required for the system to function properly. If set to Y, all projects would be forced to execute this migration when encountered. A value of N means that the migration is optional, and it would be skipped for any projects that do not list it within their <proj>_config_ready.dat file.
Config Required	Whether or not configuration is required by a project for the system to function properly. This value is set to Y whenever a change is made that requires configuration work. For instance, if a new required column is added to a configuration table, the population of this new column properly is the domain of the project engineer, not the developer. Setting this field to Y will flag to all project engineers that this migration requires their attention before the migration can be executed. The specific instructions for configuration migration must be documented in the PR's Migration section in gnats. Project engineers signify that the configuration has been examined and completed by added this migration PR to the <proj>_config_ready.dat file.
Script Exists	Indicates whether a script exists for the migration. For example, if a script exists for PR 19254, then there is a script pr19254_migration.ces that performs the migration. Not all migrations involve explicit scripts. As an example, a configuration table change would normally not require a migration. However, if it is important that a new configuration column be properly populated, this must be flagged for project engineers. This is done by adding the PR to pr_migration.dat, setting Config Required to Y and Script Exists to N. Even though there is no migration script, the migration process will not proceed until the project engineer has signified that the configuration is complete by adding the PR to the <proj>_config_ready.dat file.
Config Type	Describes the type of configuration change. Valid values are: • config_sql - A configuration SQL file has changed. • schema_sql - A schema SQL file has changed. • retain_sql - A retain SQL file has changed. • core_sql - A core (required) data SQL file has changed. • data - Model (facilities) data is being migrated. • app_defaults - New or obsolete application default options. • map_rebuild - The migration script will regenerate map files. • metafile_rebuild - The script will regenerate all map metafiles. • service_restart - Services must be restarted. • environment_restart - All user environments must be restarted.

Correcting Warnings and Errors

The table below shows the corrections for some possible errors you might receive when running the ces_apply_migrations.ces script.

Warning	Remedy
WARNING THE FOLLOWING MIGRATIONS NEED CONFIGURATION PR_NUMBER RELEASE_PATCH	This warning is displayed when migrations requiring manual changes are found. To determine the necessary changes, refer to the corresponding file in the $NMS_HOME/migration/manual directory. After making the manual changes, add the PR number to the $NMS_CONFIG/migration/<project>_config_ready.dat file.
DATABASE RELEASE LEVEL IS GREATER THAN SOURCE RELEASE LEVEL MIGRATING BACKWARDS NOT SUPPORTED	This error indicates that the schema level of the database is greater than the runtime executables that are being used. You can return to a prior release if you execute the ces_setup.ces script with the -clean command line option and perform a model build. You should not return to a prior release without running a ces_setup.ces -clean and a model build, for there may be unresolved problems that could cause system instability.

Applying Custom Migrations for Oracle Utilities Network Management System Integrators

When configuring Oracle Utilities Network Management System for a utility, integrators may wish to add custom migrations in addition to product migrations.

Process Overview

- Custom migrations are entered in an XML file (`utility_migrations.xml`) that is saved to `$NMS_CONFIG`.
- Migrations are applied by running `ces_setup.ces` or when you run `ces_apply.migrations.ces` manually. (Both scripts check for the existence of the `utility_migrations.xml` file in `$NMS_CONFIG`.)

Adding Custom Migrations to the Utility Migrations XML File

The utility_migrations.xml file has the following structure:

```
<utility_migration>
    <migration>
        <base_name>migration1</base_name>
        <before>...</before>
    </migration>
    <migration>
        <base_name>migration2</base_name>
        <before>...</before>
    </migration>
    .
    .
    .
</utility_migration>
```

- The **`<base_name>`** element contains the name of the migration, without any suffix, found either in `$NMS_HOME/migrations/sql` or `$NMS_HOME/migrations/scripts`. The **`<base_name>`** must be 32 alphanumeric characters or less and must be unique among the custom migrations.
- The **`<before>`** element must have one of the following values:

Value	Description
T	True, if the migration is to be run before product migrations.
F	False, if the migration is to be run after product migrations.

Note: Do not add a migrations to the xml file unless it actually exists in the `$NMS_HOME/migration` directories.

UTILITY_MIGRATIONS Database Table

The migrations found in the `$NMS_CONFIG/utility_migrations.xml` file are stored in the `UTILITY_MIGRATIONS` database table. The schema of that table is:

```
create table UTILITY_MIGRATIONS (
             stamp TIMESTAMP not null,
             base_name VARCHAR2(32) unique not null,
             before VARCHAR2(1) not null,
             applied VARCHAR2(1) not null,
             status VARCHAR2(32) not null)
```

The migrations stored in this table may be listed by running the command:

```
utility-apply-migrations.py list
```

The utility-apply-migrations.py script operates on ***.sql** files and **`*.ces`** scripts found in `$NMS_HOME/migrations`.

Chapter 10

Troubleshooting and Support

If you experience problems with your Oracle Utilities Network Management System, there are a number of tools and resources available to help you identify and resolve problems. These include log files, core files, Knowledge Management Documents available on My Oracle Support, and Oracle Customer Support.

This chapter includes the following topics:

- **Troubleshooting an Issue**
  - **Evaluating System Status**
  - **Examining Log Files**
  - **Examining Core Files**
  - **Identifying Memory Leaks with monitor_ps_sizes.ces**
  - **Identifying Network Latency Issues**
  - **Other Troubleshooting Utilities**
- **Oracle Support Information**
  - **Support Knowledgebase**
  - **Contacting Oracle Support**

Troubleshooting an Issue

Evaluating System Status

A good first diagnosis is to run the Unix **top** command or equivalent (**topas** on AIX; **prstat** on Solaris). This will display information such as what processes are running, current memory usage, and free memory.

Examining Log Files

Log files are the best tools for tracking down the source of a problem because very seldom does something crash or behave strangely without an entry being logged. Before reporting an issue to Oracle Customer Support, it is important to review log files for critical information that may help Oracle Customer Support solve your problem.

There are several logs that are especially useful for troubleshooting issues with NMS implementations; these include services logs and PID logs. The sections that follow describe important troubleshooting logs.

Oracle Utilities Network Management System Log Files

Application log files are located in the directory specified by the **CES_LOG_DIR** environment variable, which is defined in the .nmsrc file.

Note: by default, CES_LOG_DIR is set as $NMS_HOME/logs

- There will be one log file in this directory for each actively running service.
- After a process has been stopped and restarted, the old log file for that particular server is moved to the old_log subdirectory within the CES_LOG_DIR directory.
- After the number of days specified in $CES_DAYS_TO_LOG, old log files for a given process in the $CES_LOG_DIR/old_log directory will be purged on the next attempt to start that process. The default for CES_DAYS_TO_LOG is 7 (days). Thus, old logs will only be retained for 1 week by default.

Service Logs

Looking for DBService errors is a common starting place in determining if the problem is a database issue or a services issue. DBService errors can appear in DBService, TCDBService, and MBDBService or some other *DBService, depending upon which service is having a problem interacting with the database.

If a particular service cores, Customer Support will want to know if the service has any error messages in the log file right before it failed. The most relevant portion of the log is the text concerning what happened right before the dump. Often, there are important messages explaining why the service exited.

Another key service log is the SMService log. This log records if/when SMService attempts to restart other services.

Oracle Utilities Network Management System Log File Naming Conventions

Within the log directory, the following naming conventions apply:

- There is one log file for each Service actively executing on the server. Service logs are named <Service Name>.<date>.<time>.log. Example log files would be:

```
DBService.2010052898.111721.log
DDService.20100528.111800.log
```

Trimming and Archiving Application Oracle Utilities Network Management System Log Files

As log files grow, they generally need to be removed or archived. When determining the maximum size and content of log files, consider your company's needs:

- If accounting files need to be kept for an audit, a larger log file is justifiable. Backups of those files might even be in order.
- After the number of days specified in $CES_DAYS_TO_LOG (environment variable), the old log files for a given process in the $CES_LOG_DIR/old_log directory will be purged on the next attempt to start that process. The default for $CES_DAYS_TO_LOG is 7 (days). Thus, old logs will only be retained for 1 week by default.

Issues like these should be carefully assessed, and you should develop a policy around your company's specific needs.

PID Logs

PID logs are files with an integer value suffixed by .log. When they are generated, they also create a <pid>.out file. The .out file is unnecessary and can be removed. <pid> logs are generated in one of two ways.

- **cmd snapshot command**. This will create <pid> logs for all Isis processes currently running, whether they are services or tools. They appear in the following locations: services will appear in the `$CES_LOG_DIR/run.<service>` directory of the user that starts services. Tools will appear in the directory where ceslogin was started (typically the HOME directory of the user). If a tool is started from the command line, it will appear in the directory where the tool was started.
- **kill -usr2 <pid>**. This will NOT actually kill the tool.It will send a signal to the processs which will create a <pid>.log for that one PID, however.

  Note: You can do this multiple times, and the logs will append additional dumps into the same log file as long as the process continues to run. It will not remove or replace logs upon additional snapshots of the same process. Customer Service recommends that these logs be cleaned up upon the end of investigating an issue.

Java Application Server Log Files

The WebLogic server log files are written to the following location:

- `BEA_HOME/user_projects/DOMAIN_NAME`

  where:

  - BEA_HOME - Oracle WebLogic Server installation directory
  - DOMAIN_NAME - WebLogic domain name used for Oracle Utilities Network Management System
  - SERVER_NAME - WebLogic server name used for Oracle Utilities Network Management System

Retrieving EJB Timing Logs and Statistics from WebLogic

There are two options for retrieving EJB timing statistics. The first keeps track of the count, max, min, total, and average response times. It logs the usage in 5 minute intervals. It puts a minimal load on the system, so it can be left on at all times, if desired.

The following is an example of the information returned (times are in milliseconds):

```
First timestamp: Mon Nov 14 16:04:13 CST 2011
Last timestamp: Mon Nov 14 17:59:13 CST 2011

     Sum    Count   Avg   Min    Max  Method
 1006264    14465    69     0   7072  PublisherBean.messageHandlerLocal
 1005608    14465    69     0   7072  PublisherBean.messageHandler
 1004780    14465    69     0   7072  PublisherBean.internalMessageHandler
  992921      380  2612     0   7010  ViewerBean.flmUpdateEventHandler
   21275   147067     0     0      9  PublisherBean.getLocalPublishedMessages
   19849   208070     0     0      9  PublisherBean.getPublishedMessages
   12139       24   505   505    505  ViewerBean.getLayers
   11005      574    19     3    143  Session.executeSQLQuery
    8439       72   117     0    351  ViewerBean.getInheritanceMapping
    5006       24   208   208    208  CrewOperations.getFieldLengths
    4954       39   127     3    222  Session.getAlias
    4952       39   126     3    222  Session.getAliases
    4669       24   194   194    194  ViewerBean.getPartitions
    4457   208822     0     0      0  PublisherBean.isOnline
    2982       24   124   124    124  SwmanServiceImpl.init
...
```

The second option enables logging each EJB call with a timestamp. Since this method can generate a significant amount of logging information, it should not be enabled unless needed. Here is an example of the log:

```
2011-11-14 16:05:44 nms1 PublisherBean.getLocalPublishedMessages 0
2011-11-14 16:05:45 nms1 PublisherBean.isOnline 0
2011-11-14 16:05:45 nms1 PublisherBean.getPublishedMessages 0
2011-11-14 16:05:45 nms1 PublisherBean.getLocalPublishedMessages 0
2011-11-14 16:05:45 nms1 PublisherBean.isOnline 0
2011-11-14 16:05:45 nms1 PublisherBean.getPublishedMessages 0
...
```

Enabling Statistics

To enable statistics, cd to the WebLogic domain bin directory and run the following (replacing the username, password, url and server name); the second line should be entered all on the same line:

```
. ./setDomainEnv.sh
```

```
java weblogic.WLST scripts/configure_statistics.py weblogic weblogic1
t3://localhost:7001 server_name
```

There should not be any errors. You should see output that ends with the following:

```
Activation completed
EJB Statistics have been successfully enabled.

Exiting WebLogic Scripting Tool.
```

If there are errors, recheck the server name and connection information.

Enabling EJB Logging

To log each ejb call, add or uncomment out the line in $NMS_CONFIG/jconfig/build.properties:

```
weblogic.ejb_logging = true
```

Then run:

```
 nms-install-config --java
```

Then install this by running:

```
nms-install-config --java
```

Accessing Statistics

The statistics are accessed from the nms server (not necessarily the WebLogic server).

To access the stats, use the following script (replacing the user name, password, and url with your system):

```
wls_stats_summary.ces weblogic weblogic1 http://localhost:7001 2011-
11-01 13:00 2011-11-02 13:00
```

The date/times are the start and end times that you are interested in. If only one date/time is entered, it will go to the end of the log. If no times are given, than the entire log will be transferred. The date may be omitted to get the information from the current day.

Accessing the Log

To access the logs, use this script, which takes the same parameters as above

```
wls_stats_dump.ces weblogic weblogic1 http://localhost:7001 2011-11-01
13:00 2011-11-02 13:00
```

Configuring the Retention of Statistics and Logs

This is managed by WebLogic. It can be set up to either keep a certain size of the archive, or to keep a certain number of hours of information. The statistics information is stored in the HarvestedDataArchive, and the logs are stored in the EventsDataArchive. See the Weblogic documentation for more information:

http://docs.oracle.com/cd/E11035_01/wls100/wldf_configuring/config_diag_archives.html#wp1069508

Java Client Application Logs

Java client applications executing on a user's desktop by default do not generate log files. To obtain their output (error messages, exceptions and debug information) the Windows Java console can be used but it must first be enabled.

Use following steps to enable the Java console in Windows:

1. Open the Control Panel (Start -> Settings -> Control Panel).
2. Open the Java Control Panel by double-clicking on the Java icon in the Control Panel.
3. Select the Advanced tab of the Java Control Panel.
4. Set Java console parameter to 'Show console' (Java console will be started maximized) or 'Hide console' (Java console will be started minimized).
5. Under Settings -> Debugging, enable tracing and logging. The default location for a java log is `%USERPROFILE%\Application Data\Sun\Java\Deployment\log`.
6. Press **OK.**

Isis Log Files

There are two types of Isis log files:

- An Isis startup log, which logs everything before protos is completely started, should it exit for some reason. The isisboot program starts isis (isis in turn starts protos) using the nohup command, which makes protos immune to hang-ups, like exiting the terminal after starting Isis. The startup log is called isis.log and can be found in `$NMS_HOME/etc/run_isis`. If you cannot start Isis, check this log.
- The protos log contains log information for the running protos process. This file is site-specific, and the name is based on the site number of the machine on which protos is running. The log for the protos process can be found in $CES_LOG_DIR/run.isis/<site #>.logdir/<site #>_protos.<date>.<time>.log.
- When Isis is restarted, the old log files will be archived into the $CES_LOG_DIR/run.isis/<site#>.logdir/old_log directory. The will be automatically removed after $CES_DAYS_TO_LOG if/when Isis is restarted.

Oracle RDBMS Log Files

Many times, there is an error in an Application log file that points to some sort of database problem. DBService may log that at a certain time the database was unavailable to answer queries. Look in the database logs to find the answer. These logs can alert you to problems with the RDBMS configuration, software, and operations. Other instances of a dbservice (TCDBService, PFDBServicee, MBDBService may also have configured and running. Each of these should be reviewed for errors.

Refer to the Oracle RDBMS documentation for locations and instructions for viewing Oracle RDBMS logs.

Operating System Log Files

Another place to look for problems is in the operating system logs. Refer to the operating-system-specific documentation for locations and instructions for viewing operating system logs (generally various forms of syslog - like /var/log/messages for Linux).

It is generally recommended that syslog be turned on for a production system. In particular, the Oracle Utilities Network Management System uses the syslog to track fatal errors and log the start/stop time of every Oracle Utilities Network Management System-specific Unix process.

Entries like the following can be useful when trying to track down which application binary a particular Unix process ID belongs to:

- May 30 12:47:57 msp-pelin01 CES::corbagateway[26346]: my_address = (2/7:26346.0)
- May 30 12:48:00 msp-pelin01 CES::corbagateway[26346]: **INFO*** [corbagateway-26346] for [msp-pelin01] exiting....

Using the Action Command to Start a New Log File

There is also a feature that uses the Action command to start a new log file without stopping anything. This can be very useful in isolating a portion of the log file when recreating a problem. The command is:

```
Action any.<NMS_ISIS_process_name> relog
```

For example: `Action any.JMService relog`

The Action command can also be used to turn debug on and off for services or tools. This can also be used with the relog feature to better isolate debug for a particular user scenario.

The following command will turn debug on:

```
Action any.<service> debug 1
```

The following command will turn debug off:

```
Action any.<service> debug 0
```

Examining Core Files

On Unix, if a process has either committed an error or over-taxed the system resources, the O/S will kill it rather than letting it take down the operating system. When this happens, the operating system dumps the contents of the memory occupied by the process into a file named “core.” These files can sometimes be analyzed to better understand the reason for the failure.

Normally, you should question the production of a core file to see if there are any extraneous reasons why the O/S is dumping a process. If you do not find anything, retrieve the core file and analyze it.

> **Note:** see **Core File Naming Configuration** on page 3-5 for OS specific information about core file naming.

Core files are located in the `CES_LOG_DIR/run.<service>` directory in the username that started services, or in the directory where a tool was started (usually the home directory of the user).

After performing a **`kill -USR2`** on a hung process, it can be useful to follow with **`kill -abrt`** ***`<pid>`***

This will cause the process to dump core and the process will be dead.

> **Note**: Always use **`kill -USR2`** before **`kill -abrt`** because the `-abrt` option terminates the process. Make sure it is ok to terminate the process before attempting **`kill -abrt`**

The command **`file core`** will generally (depending on the operating system involved) identify which process generated the core. Later core files can overwrite earlier core files. Renaming the core file to something like **`core.<process>`** can prevent this.

SMService can be set up to automatically find, rename, and consolidate core files into a single directory (`$CES_LOG_DIR/SavedCores` by default). You can change what happens to core files captured by SMService by modifying the `sms_core_save.ces` script.

When a tool or service cores, the investigation is helped by sending the stack trace in the incident report. A stack trace can be generated using the dbx (Solaris and AIX) or gdb (Linux) tool. The syntax is as follows:

Solaris:

```
dbx <path to binary directory> <path to corefile>
```

AIX:

```
dbx -d 10000 <path to binary directory> <path to corefile>
```

Linux :

```
gbd <path to binary directory> <path to corefile>
```

For example:

```
dbx $CES_HOME/bin/JMService ~/run.JMService/core
```

Press the space bar until you get a prompt and then type the following commands:

Solaris:

```
where
threads
dump
regs
quit
```

AIX:

```
where
thread
dump
registers
quit
```

Linux:

```
where
info threads
info locals
info all-reg
thread apply all where
```

Then include the results of these commands when you report the incident.

Searching for Core Files

To search for core files, complete these steps:

1. Search for core files with the find command:

   ```
   $ find . -name core* -exec ls -l {} \;
   ```

   Expected result:

   ```
   -rw------- 1 ces users 32216692 Oct 15 16:05 ./core
   ```

   This executes an "ls -l" on any files found in the tree starting from the current working directory. This should be done from the $NMS_HOME directory and (if it differs from $NMS_HOME) the $HOME directory.

   If a service cores, the core file can be found in the `$CES_LOG_DIR/SavedCores` or (if SMService failed or is not configured with a CoreScript to detect and/or move the core file) the `$CES_LOG_DIR/run.<service> directory`. Note that SMService will rename a service core file to `<hostname>-<service>-<date>.<time>.core` to minimize the chance of core files overwriting each other.

2. Type the following to determine where a core file came from:

   ```
   $ file ./core
   ```

   Below is a sample result from an AIX server:

   ```
   core: AIX core file fulldump 64-bit, JMService - received SIGBUS
   ```

   The core file referenced above is the result of a JMService core dump. The output gives:

   - the file name (which is always "core"),
   - which program/process the file came from (JMService), and
   - optionally, the message that the program received from the OS (SIGBUS).

3. Generally the most useful thing you can do is to identify what is called the core stack trace--the specific functions that were called (in order) leading up to the violation that caused the operating system to generate the core file. The stack trace is often a useful piece of information that, if available, should be captured for later analysis. Details on navigating a core trace can be found later in this document.

4. Use the strings command to get some more information out of the file, if possible. Type:

   ```
   $ strings core | head
   ```

   Sometimes the messages returned, such as "Out of memory" or "I/O error," give an idea of what might have happened.

Identifying Memory Leaks with monitor_ps_sizes.ces

The monitor_ps_sizes.ces script monitors the size of processes to identify potential leaks. It performs periodic snapshots of all running processes and warns the user of any processes that have grown greater than the specified size. It supports the following command-line options:

Option	Description
-n <program names>	A comma-separated list of program names to monitor
-l <line number>	The line number that specifies the stable size in the process-size log file. Default: 3 (line numbers begin counting with 1)
-l <line number>	The line number that specifies the stable size in the process-size log file. Default: 3 (line numbers begin counting with 1)
-p <number>	The number of seconds to wait between snapshots. Default: 3600 (seconds)
-g <number>	The growth factor that triggers a report. Default: 1.75 (floating point numbers greater than 1 are valid)
-R <number>	The minimum process size that can be reported. Default: 5000 (units reported by ps)
-G <number>	A warning about a process is guaranteed to be generated if the process exceeds this size. Default: 40000 (units reported by ps)
-P <number>	The minimum number of seconds to wait between warnings. Default: 0 (seconds)
-O <number>	The maximum number of seconds to retain log files.Default: 172800 (seconds) if 0, old log files are not erased.
-u <email names>	A comma-separated list of users to email when there are processes warnings. Default: no email sent.
-s <email subject line>	The subject line to use to title email warnings about processes that are too big. Default: "process size warning for prod_model"
-a <command>	Command to perform on process when generating a warning. You can pass the program's name and/or PID via #PID# and #PROGRAM#
-A	Log the command's output

For example, to monitor JMService and MTService for user 'nms' when either gets larger than 500 meg or grows by 10%, use:

```
monitor_ps_sizes.ces -n MTService,JMService -f nms -p 30 -R 500000 -g 1.1
```

Identifying Network Latency Issues

The NMS Network Connectivity Utility, which calculates network latency, is useful in diagnosing performance issues. The utility can be launched in a browser using the following URL convention:

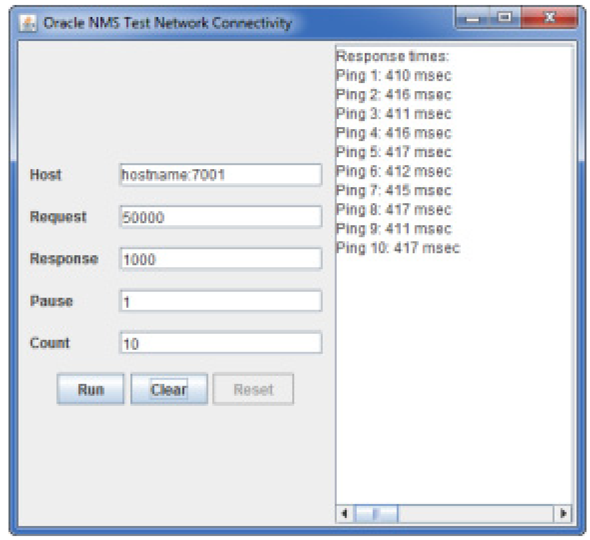
`http://hostname:7001/nms/Ping.jnlp`

substituting the hostname and port with the host and port of your managed server.

The Oracle NMS Test Network Connectivity tool allows you to set parameters for testing network response:

.**Fields**

- **Host:** the host to connect to; this field is prepopulated from the server.
- **Request:** sets the number of additional bytes to send as part of the request.
- **Response:** sets the number of additional bytes to send as part of the response.
- **Pause:** sets the number of seconds between requests.
- **Count:** sets the number of times to run the ping.



Action Buttons

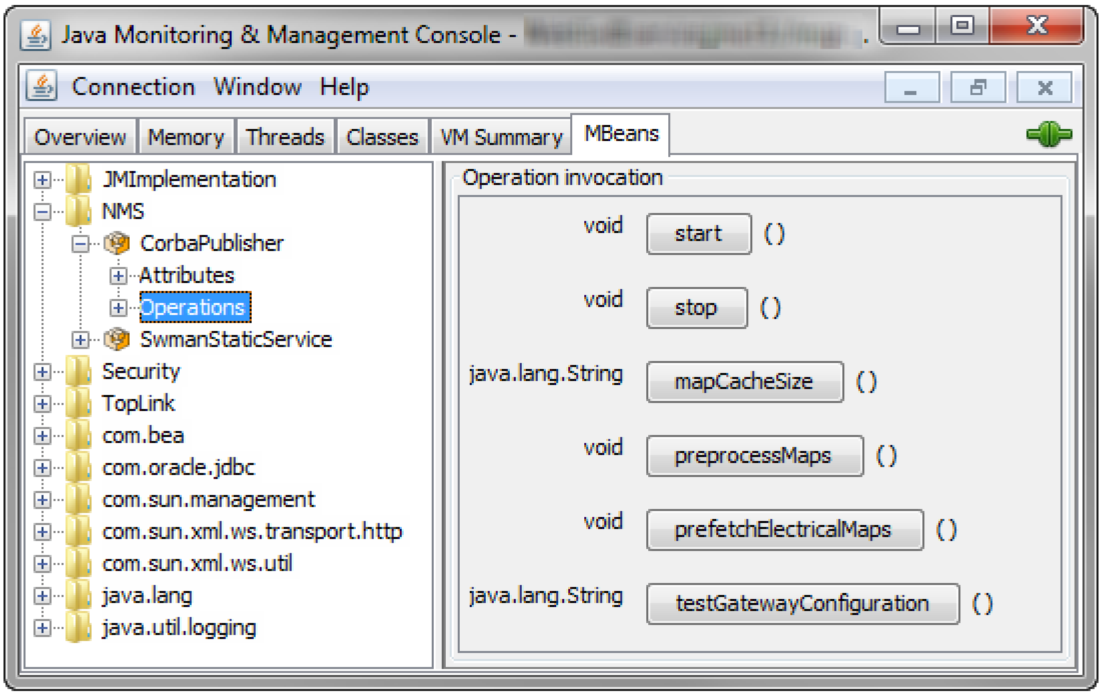
- **Run**: click to run the ping.
- **Clear**: click to clear the listing of pings in the **Response times** pane.
- **Reset**: click to abort a running ping request.

Testing the Web Gateway

If you are experiencing connection problems when trying to start the Web Gateway, perform the following steps to check that it is configured properly.

1. Open the Java Monitoring & Management Console by creating a new JConsole connection for the user running the Web Gateway.

2. Click the MBeans tab.



3. In the tree view on the left, select **Operations** under **NMS>CorbaPublisher**.

4. Click **testgatewayConfiguration**.

The **Operation return value** dialog will open, which should report any configuration errors that it finds.

Other Troubleshooting Utilities

Using the JMS API Command Line Utility to Manually Change a Job

The JMS API command line utility (jms_api) provides a restricted set of options to modify an event when the event cannot be changed with the NMS user interface. It is primarily intended for cleaning up stranded events or other issues where normal NMS functionality will not work. Indiscriminate use of jms_api for frequent/high volume activity in conjunction with NMS operation can have a negative performance impact on NMS and is not recommended

Standard Usage:

```
# jms_api <option> <event>
```

where

- <option> is the jms_api option
- <event> is an event handle of the form 800.<event#> (e.g., 800.10257)

- To return the jms_api usage options and arguments:

  ```
  # jms_api
  ```

- To complete an event:

  ```
  # jms_api complete <event> "<comments>"
  ```

  Note: does not allow an RDO still affecting customers to be completed.

- To cancel an event:

  ```
  # jms_api cancel <event> "<comments>"
  ```

  Note: does not allow an RDO still affecting customers to be canceled.

- To complete a Master Switching Job or Planned Outage:

  ```
  # jms_api swplan_complete <event> "<comments>"
  ```

  Note: does not allow an active Planned Outage or a Master Switching Job with active Planned Outage(s) to be completed

- To cancel ("reschedule") a Master Switching Job:

  ```
  # jms_api swplan_cancel <event> "<comments>"
  ```

- To remove association between event and switch sheet:

  ```
  # jms_api remove_assoc <event> <sheet.handle>
  ```

- To change the estimated restore time of a job:

  ```
  # jms_api set_est_rest_time <event> <time>
  ```

  Note: <*time*> must be a valid ISO-8601 date/time string (*e.g.*, *2012-02-27T15:30*).

- To set the external id of a job:

```
# jms_api set_external_id <event> <value>
```

- To set the customers out for a job:

```
# jms_api set_cust_out <event> <value>
```

- To set the trouble code of a job:

```
# jms_api set_trouble_code <event> <value>
```

  Note: does not modify the calls on a job; *<value>* must be a valid numeric trouble code.

- Alternative API for completing an event

```
# jms_api complete2 <event>
```

  Note: does very little validation, but does require a state transition to be configured with act_type='SRS' and act_val=14

Oracle Support Information

Support Knowledgebase

Additional troubleshooting information can be found on My Oracle Support at:

`http://support.oracle.com`

Contacting Oracle Support

For support please contact Oracle Support at:

`http://www.oracle.com/support/index.html`

Chapter 11

Setting Up Oracle Business Intelligence

This chapter describes how to set up Oracle Business Intelligence for Utilities (BI) for use with NMS. It includes the following topics:

- **Installing BI**
- **Installing NMS BI Extractors**
- **Running NMS BI Extractors**
- **Migrating from Performance Mart to BI**

Installing BI

Installation of the BI component is covered in a separate installation guide that comes with the BI Media Pack download.

> **Note**: If you are upgrading from a previous Performance Mart data warehouse, please reference the **Migrating from Performance Mart to BI** on page 11-4 for details on the upgrade process.

BI must be properly installed before you can perform the remaining procedures described in this chapter.

Installing NMS BI Extractors

CES_PARAMETERS Configuration

BI version 2.4.0 introduced schema changes that is handled in NMS by setting a BI version attribute in the CES_PARAMETERS database table. The attribute, **`BI_VERSION`**, is populated with an app of '`NMS`' and the BI version value.

For BI 2.4, for example, the *`<project>`*`_parameters.sql` file would be need the following `INSERT` statement:

```
INSERT INTO CES_PARAMETERS (APP, ATTRIB, VALUE) VALUES
  ('NMS',
   'BI_VERSION',
   '2.4');
```

If `BI_VERSION` is not set in the *`<project>`*`_parameters.sql` file, NMS defaults to BI 2.2.

Environment Variable Configuration

If extracting from separate NMS environments into a common BI environment, edit the `.nmsrc` to uncomment the export line for `CES_BI_DATA_SOURCE` by removing the leading `#`:

```
export CES_BI_DATA_SOURCE=4
```

The default value is 4. For more information, see the **CES_BI_DATA_SOURCE** definition on page 11-21.

Extractor Installation

To install the NMS BI extractors, run the install_business_intelligence script. Once this script has been run, use the refresh_business_intelligence script for any subsequent configuration and schema changes. This script generates a log file, create_bi_extractors.log, which lists any errors.

Running NMS BI Extractors

Extractor Overview

This section explains the extractor scripts, which should be configured to run in scheduled cron jobs. Each of these scripts creates a set of extract files, which are direct queries from NMS database views. The mapping of these views to BI database tables is documented with Oracle database comments. Access them by performing the following query in the NMS database:

```
SELECT * FROM user_tab_comments WHERE table_name LIKE '%MODIFY_V' AND
comments IS NOT NULL;
```

bi_common_extractor

This extracts the model-related information like devices and control zones. This script is designed to be run daily, after model changes.

bi_event_extractor

This extracts completed outages and call information. This script is designed to be run daily.

bi_customer_extractor

This extracts customer information. This script is designed to be run daily, after customer data changes.

bi_feeder_extractor

This extracts feeder load information. This script is designed to be run hourly to report average hourly loads.

bi_switch_extractor

This extracts planned switching information. This script is designed to be run daily to report switching activity.

nrt_extractor

This extracts current outage, call, and storm information. This script is designed to be run 3 to 4 times an hour, throughout the day.

Notes about Extractors

These scripts create extract *.dat* and *.ctl* files in bi_extract_dir database directory; the default is $HOME/extract unless NMS_BI_DIRECTORY environment variable is set. These files will be read by the Business Intelligence import process.

Each script generates a log file named, for example, bi_common_extractor.log, which should list any errors.

The bi_feeder_extractor should not be run more frequently than once an hour, and the nrt_extractor can be scheduled to run every 15 minutes. The order that these two extractors run does not matter. To schedule the daily extracts, bi_common_extractor, bi_event_extractor, bi_switch_extractor, and bi_customer_extractor should run in the following order:

1. bi_event_extractor
2. bi_switch_extractor
3. bi_common_extractor
4. bi_customer_extractor

Importing NMS Extract Files

The extract files created by running the NMS extractors must be moved to the directory specified in the EditFP.tcl script that is executed when BI is installed. There are various mechanisms that a System Administrator can use to copy these files, including FTP scripts and Cross Mounting hard drives. However, Oracle does not provide any scripts to copy extract files, so a customer is responsible for putting these in place.

Once the extract files have been copied to the appropriate import directory, the NMS Process Flows described in the NMS Facts and Dimensions chapter of the BI documentation need to be run to load the data contained in the files. The process flows corresponding to each extract program is documented in this chapter, and the import process and how to automate it is described in the Oracle Warehouse Builder chapter of the BI documentation.

After importing the data, then the various NMS zones and portals that a customer has created can be opened or refreshed to view the NMS data in BI.

The next steps are a method to import from the extracted files using a function call in sqlplus.

1. Install the Function NMS_EXEC_WF_FNC to Execute Process Flows from SQLPLUS
   - For 10g, install the script nms_exec_wf_fnc_10.sql

   ```
   sqlplus birepownuser/birepownpasswd@birepown_instance
   < nms_exec_wf_fnc_10.sql > nms_exec_wf_fnc_10.sql.log
   ```

   - For 11g install the script nms_exec_wf_fnc_11.sql

   ```
   sqlplus birepownuser/birepownpasswd@birepown_instance
   < nms_exec_wf_fnc_11.sql > nms_exec_wf_fnc_11.sql.log
   ```

2. Make sure the following environment variables are set:
   - BIREPOWN_USER - BI Repository User
   - BIREPOWN_PASSWD - BI Repository Password
   - BIREPOWN_INSTANCE - SQL*Net connection to the BI Repository Database

3. Run the Import Into DWADM Schema from the Extracted Files. For the daily extracts, set the following scripts to run on schedule after the entire daily extract has run, in the following order:

    1. bi_customer_import - call this script after the bi_customer_extractor runs.
    2. bi_common_import - call this script after the bi_common_extractor runs.
    3. bi_switch_import - call this script after the bi_switch_extractor runs.
    4. bi_event_import - call this script after the bi_event_extractor runs.

    For the other two extracts, set the import to run after the extract has taken place:

    - bi_feeder_import - call this script after the bi_feeder_extractor runs.
    - bi_nrt_import - call this script after the nrt_extractor runs.

Migrating from Performance Mart to BI

This section provides an overview of the schema differences that you must be aware of when migrating from Performance Mart to BI.

In version 1.9 of the NMS, the Performance Mart and Executive Dashboard modules were replaced with Oracle Business Intelligence version 2.2.1. This section describes the differences between the two products, how to migrate an existing 1.7.10 Performance Mart database to Oracle Business Intelligence, and provides some guidelines on how to easily migrate existing reports to run against the Oracle Business Intelligence database.

For information not covered in this document, the Oracle Business Intelligence documentation is available for all supported releases, including the Oracle Utilities Network Management System Facts and Dimensions chapter that describes the schema and extraction processes that will be covered in this guide.

Schema Differences

The Oracle Business Intelligence database naming system is different than the Performance Mart schema, so every NMS object has a new name. Also, Oracle Business Intelligence utilizes a very strict star-schema approach, so many of the Command Centricity foreign key relationships do not exist.

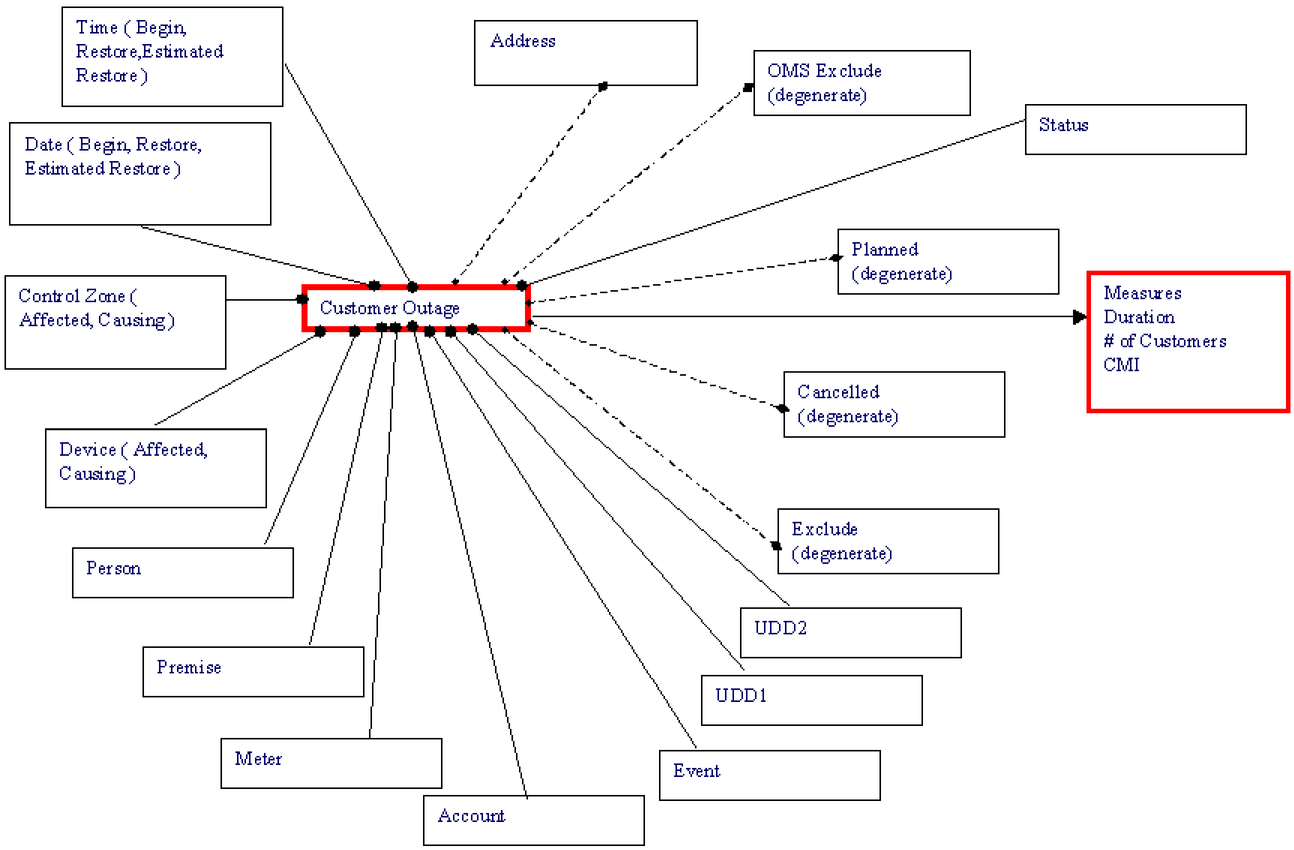
Performance Mart Schema

The Performance Mart schema is a hybrid star-schema/relational model that was convenient for use with Executive Dashboard, and detail trouble reporting.

Oracle Business Intelligence Schema

Unlike Performance Mart, the Oracle Business Intelligence Schema utilizes exclusively a Star schema representation. This enables the Oracle Business Intelligence framework to efficiently create queries against the database tables, and allows for an efficient generic load process.

The following figure shows the star schema diagram for the Customer Outage Fact. This fact corresponds with the SERVICE_POINT_SUPPLY_NODES table in the Performance Mart schema. If you compare the relationships here with the relationships above, you will notice a lot more foreign keys in this document, but nothing related more than one step away from the basic fact table.



The other major difference between Performance Mart and Oracle Business Intelligence is the use of generic field names in the tables. This is done to allow different customers to extract different fields without having to change the user interface or extractor code. For example, the Device information is stored in the DEVICE_DETAILS table in Performance Mart and in the CD_DEVICE table in Oracle Business Intelligence.

The following table lists the fields in each table and how they map from one to another

Device_Details	CD_Device
DV_CLS	SRC_DEVICE_CLS
DV_IDX	SRC_DEVICE_IDX
DV_CODE	DEVICE_NAME
DV_VOLTAGE	Unmapped
DV_TYPE	DEVICE_CLASS_CD
DV_DESC	DEVICE_CLASS_DESCR
DV_ACTIVE	Unmapped
	DEVICE_TYPE_CD
	DEVICE_TYPE_DESCR
	UDF1_CD
	UDF1_DESCR

Device_Details	CD_Device
	UDF2_CD
	UDF2_DESCR
	UDF3_CD
	UDF3_DESCR
	UDF4_CD
	UDF4_DESCR
	UDF5_CD
	UDF5_DESCR
	UDF6_CD
	UDF6_DESCR
	UDF7_CD
	UDF7_DESCR
	UDF8_CD
	UDF8_DESCR
	UDF9_CD
	UDF9_DESCR
	UDF10_CD
	UDF10_DESCR

Performance Mart to BI Mapping

The following tables show how the default migration routine will move data from Performance Mart tables to Oracle Business Intelligence tables. Performance Mart tables not listed here will not be migrated to Oracle Business Intelligence. Project configuration changes done during the actual migration can change how these columns are migrated, so this list should not be used as a definitive guide to a specific project implementation.

CU_SERVICE_LOCATION_DETAILS

The data in the CU_SERVICE_LOCATION_DETAILS table is migrated to three different BI tables: CD_ACCT, CD_ADDR and CD_PREM. The following table shows which fields go into which table. The CU_SERV_LOC_KEY is used as the primary key in each of these tables.

CU_SERVICE_LOCATION_DETAILS Field	BI Table Name	BI Field Name
cu_serv_loc_key	cd_acct	acct_key
cu_serv_account_number	cd_acct	src_acct_id
cu_serv_loc_id	cd_acct	acct_info

CU_SERVICE_LOCATION_DETAILS Field	BI Table Name	BI Field Name
record_birth_time	cd_addr	eff_start_dttm
record_death_time	cd_addr	eff_end_dttm
cu_serv_loc_key	cd_addr	addr_key
cu_serv_addr_1	cd_addr	addr_line1
cu_serv_addr_2	cd_addr	addr_line3
cu_serv_addr_3	cd_addr	addr_line4
cu_serv_city	cd_addr	udf1_cd, udf1_descr
cu_serv_postcode_1	cd_addr	udf3_cd
cu_serv_postcode_1 || cu_serv_postcode_2	cd_addr	udf3_descr
cu_serv_state	cd_addr	udf4_cd, udf4_descr
cu_serv_loc_id	cd_addr	src_addr_id
record_birth_time	cd_addr	eff_start_dttm
record_death_time	cd_addr	eff_end_dttm
cu_serv_loc_key	cd_prem	prem_key
cu_serv_loc_id	cd_prem	src_prem_id
cu_serv_type	cd_prem	udf2_cd, udf2_descr
cu_serv_life_support	cd_prem	udf3_cd, udf3_descr
cu_serv_c_priority	cd_prem	udf6_cd, udf6_descr
cu_serv_d_priority	cd_prem	udf7_cd, udf7_descr
cu_serv_k_priority	cd_prem	udf8_cd, udf8_descr
record_birth_time	cd_addr	eff_start_dttm
record_death_time	cd_addr	eff_end_dttm

CU_CUSTOMER_DETAILS

The data in the CU_CUSTOMER_DETAILS table is migrated to the CD_PER table in BI. The CU_CUST_KEY is used as the primary key in this table.

CU_CUSTOMER_DETAILS Field	BI Table Name	BI Field Name
cu_cust_key	cd_per	per_key
cu_cust_id	cd_per	src_per_id
cu_cust_name	cd_per	per_name, per_info
cu_cust_home_ac || cu_cust_home_phone	cd_per	per_phone_nbr
record_birth_time	cd_per	eff_start_dttm

CU_CUSTOMER_DETAILS Field	BI Table Name	BI Field Name
record_death_time	cd_per	eff_end_dttm

CU_METER_DETAILS

The data in the CU_METER_DETAILS table is migrated to the CD_METER table in BI. The CU_METER_KEY is used as the primary key in this table.

CU_METER_DETAILS Field	BI Table Name	BI Field Name
cu_meter_key	cd_meter	meter_key
cu_meter_id	cd_meter	src_meter_id, meter_info
record_birth_time	cd_meter	eff_start_dttm
record_death_time	cd_meter	eff_end_dttm

REPORTING_ELEMENTS - Cities

The data in the REPORTING_ELEMENTS table where the RE_TYPE = 'CITY' is migrated to the CD_CITY table in BI. The RE_KEY is used as the primary key in this table

REPORTING_ELEMENTS Field	BI Table Name	BI Field Name
re_key	cd_city	city_key
substr(re_name, 1, instr(re_name, ','))	cd_city	src_city
substr(re_name, instr(re_name, ',') + 2)	cd_city	src_state
'United States of America'	cd_city	src_country
record_birth_time	cd_city	update_dttm

REPORTING_ELEMENTS/REPORTING_HIERARCHY - Control Zones

The data in the REPORTING_ELEMENTS table where the RE_TYPE = 'CIR' is joined to the REPORTING_HIERARCHY_V view and this data is migrated to the CD_CONTROL_ZONE table in BI. The RH_KEY in the REPORTING_HIERARCHY table is used as the primary key in this table.

Performance Mart Field	BI Table Name	BI Field Name
reporting_hierarhcy_v.rh_key	cd_ctrl_zone	ctrl_zone_key
reporting_elements.re_number	cd_ctrl_zone	src_ncg_id
reporting_elements.re_type	cd_ctrl_zone	hierarchy_type
reporting_elements.re_name	cd_ctrl_zone	ctrl_zone_name
reporting_elements_1.re_number	cd_ctrl_zone	uf1_cd

Performance Mart Field	BI Table Name	BI Field Name
reporting_hierarhcy_v.level1_name	cd_ctrl_zone	udf1_descr
reporting_elements_2.re_number	cd_ctrl_zone	udf2_cd
reporting_hierarhcy_v.level2_name	cd_ctrl_zone	udf2_descr
reporting_elements_3.re_number	cd_ctrl_zone	udf3_cd
reporting_hierarhcy_v.level3_name	cd_ctrl_zone	udf3_descr
reporting_elements_4.re_number	cd_ctrl_zone	udf4_cd
reporting_hierarhcy_v.level4_name	cd_ctrl_zone	udf4_descr
reporting_elements_5.re_number	cd_ctrl_zone	udf5_cd
reporting_hierarhcy_v.level5_name	cd_ctrl_zone	udf5_descr
reporting_elements_6.re_number	cd_ctrl_zone	udf6_cd
reporting_hierarhcy_v.level6_name	cd_ctrl_zone	udf6_descr
record_birth_time	cd_ctrl_zone	update_dttm
record_death_time	cd_meter	eff_end_dttm

CREW_DETAILS

The data in the CREW_DETAILS table is migrated to the CD_CREW table in BI. The CR_KEY is used as the primary key in this table.

CREW_DETAILS	BI Table Name	BI Field Name
cr_key	cd_crew	crew_key, src_crew_id
cr_crew_code	cd_crew	crew_cd
record_birth_time	cd_crew	eff_start_dttm
record_death_time	cd_crew	eff_end_dttm

DEVICE_DETAILS

The data in the DEVICE_DETAILS table is migrated to the CD_DEVICE table in BI. The DV_KEY is used as the primary key in this table. Also, during the population, data for the Device Type fields that is not in Performance Mart is queried from the CLASSES table in the NMS database and populated into BI. If historical data does not exist for a specific class in NMS anymore, then these fields will be left blank.

DEVICE_DETAILS Field	BI Table Name	BI Field Name
dv_key	cd_device	device_key
dv_cls	cd_device	src_device_cls
dv_idx	cd_device	src_device_idx
dv_code	cd_device	device_name

DEVICE_DETAILS Field	BI Table Name	BI Field Name
dv_type	cd_device	device_class_cd
dv_desc	cd_device	device_class_descr
classes.c_type	cd_device	device_type_cd, device_type_descr
record_birth_time	cd_device	eff_start_dttm
record_death_time	cd_device	eff_end_dttm

NMS Users

No data exists in Performance Mart for NMS Users, so during the migration process, the current records in the CES_USERS table will be migrated to the CD_USER table in BI. The primary key will be populated from the SPL_USER_SEQ.NEXTVAL sequence that is normally used by the BI load process.

CES_USERS Field	BI Table Name	BI Field Name
user_name	cd_user	user_cd
full_name	cd_user	user_descr
sysdate	cd_user	eff_start_dttm
31-DEC-4000	cd_user	eff_end_dttm

Event Statuses

No data exists in Performance Mart for Event Statuses, so during the migration process, the current records in the TE_STATUSES and TE_STATUS_GROUPS tables will be migrated to the CD_USER table in BI. The primary key will be populated from the TRANS_STATUS field in the TE_STATUSES table.

NMS Field	BI Table Name	BI Field Name
te_statuses.trans_status + 1	cd_event_status	event_status_key
te_statuses.trans_status	cd_event_status	src_status
te_status_groups.description	cd_event_status	event_status_cd
te_statuses.description	cd_event_status	event_status_descr
Sysdate	cd_event_status	update_dttm

EVENT_CALL_FACTS

The data in the EVENT_CALL_FACTS table is migrated to two different BI tables, one dimension and one fact: CD_CALL_INFO and CF_RST_CALL. The following table shows which fields go into which table. The ECF_KEY is used as the primary key in each of these tables. For the BI tables below that are not CD_CALL_INFO or CF_RST_CALL, the mapping is done by using the foreign key in the CF_RST_CALL table. For example, to get the ECF_ACCOUNT_NUMBER, the CF_RST_CALL table would be joined to the CD_ACCT table by ACCT_KEY.

EVENT_CALL_FACTS Field	BI Table Name	BI Field Name
ecf_key	cd_call_info	call_info_key
ecf_incident_number	cd_call_info	src_incident_id
ecf_last_name	cd_call_info	caller_name
ecf_phone_number	cd_call_info	phone_nbr
ecf_complaint	cd_call_info	Complaint
ecf_operator_comment	cd_call_info	Comments
sysdate	cd_call_info	update_dttm
ecf_key	cf_rst_call	rst_call_key, call_info_key
ecf_incident_number	cf_rst_call	src_incident_id
e_key	cf_rst_call	event_key
ecf_account_number	cd_acct	src_acct_id
ecf_total_priority	cf_rst_call	priority_ind
ecf_called_time (Date)	cd_date	cal_dt
ecf_called_time (Time)	cd_time	src_time
ecf_user_name	cd_user	user_cd

EVENT_DETAILS

The data in the EVENT_DETAILS table is migrated to two different BI tables, one dimension and one fact: CD_EVENT and CF_RST_JOB. The EVENT_PICKLIST table is also joined to the EVENT_DETAILS table and data in this table is migrated to the CD_EVENT table. The following table shows which fields go into which table. The E_KEY is used as the primary key in each of these tables. For the BI tables below that are not either CD_EVENT or CF_RST_JOB, the mapping is done by using the foreign key in the CF_RST_JOB table. For example, to get the ECF_ACCOUNT_NUMBER, the CF_RST_CALL table would be joined to the CD_ACCT table by ACCT_KEY.

EVENT_DETAILS Field	BI Table Name	BI Field Name
e_key	cd_event	event_key
e_outage_number	cd_event	src_nbr
e_event_idx	cd_event	event_nbr
e_ops_exclude_reason	cd_event	exclude_reason
e_operator_comment	cd_event	operator_comment
e_valid_state_key	cd_event	event_state_descr

EVENT_DETAILS Field	BI Table Name	BI Field Name
e_event_status	cd_event	event_state_cd
e_street_address || ',' || e_city_state	cd_event	first_call_addr
event_picklist.remedy_om	cd_event	remedy_cd
e_trouble_code	cd_event	trouble_cd_list
e_outage_cause_selection1	cd_event	udf1_cd, udf1_descr
e_outage_cause_selection2	cd_event	udf2_cd, udf2_descr
e_outage_cause_selection3	cd_event	udf3_cd, udf3_descr
e_outage_cause_selection4	cd_event	udf4_cd, udf4_descr
e_outage_cause_selection5	cd_event	udf5_cd, udf5_descr
e_outage_cause_selection6	cd_event	udf6_cd, udf6_descr
e_outage_cause_selection7	cd_event	udf7_cd, udf7_descr
e_outage_cause_selection8	cd_event	udf8_cd, udf8_descr
e_outage_cause	cd_event	udf9_cd, udf9_descr
e_outage_cause_selection	cd_event	udf10_cd, udf10_descr
e_key	cf_rst_job	rst_job_key, event_key
e_outage_number	cf_rst_job	src_job_nbr
e_status + 1	cf_rst_job	event_status_key
e_begin_time	cf_rst_job	begin_dttm
e_completion_time	cf_rst_job	rst_dttm
e_est_restore_time (est_rst_date_key)	cd_date	cal_dt
e_est_restore_time (est_rst_time_key)	cd_time	src_time
e_ops_exclude_flag	cf_rst_job	oms_exclude_ind
e_cancel_flag	cf_rst_job	cancelled_ind
re_key	cf_rst_job	ctrl_zone_key
dv_key	cf_rst_job	device_key
e_crew_id1	cd_crew	src_crew_id
e_est_num_cust	cf_rst_job	udm1

Customer Outage

Customer Outage information is stored in three key tables in Performance Mart: SERVICE_POINT_SUPPLY_NODES, EVENT_SUPPLY_NODES and EVENT_DETAILS. Data from each of these tables as well as Customer Keys in the CUSTOMER_SERVICE_POINTS table will be migrated to the CF_CUST_RST_OUTG table in BI. The primary key will be populated from the SPL_CUST_RST_OUTG_SEQ.NEXTVAL sequence that is normally used by the BI load process.

Performance Mart Field	BI Table Name	BI Field Name
service_point_supply_nodes.e_key	cf_cust_rst_outg	event_key
customer_service_points.cu_serv_loc_key	cf_cust_rst_outg	acct_key, prem_key, addr_key
customer_service_points.cu_cust_key	cf_cust_rst_outg	per_key
customer_service_points.cu_meter_key	cf_cust_rst_outg	meter_key
service_point_supply_nodes.cu_begin_time	cf_cust_rst_outg	begin_dttm
service_point_supply_nodes.cu_completion_time	cf_cust_rst_outg	rst_dttm
event_supply_nodes.re_key	cf_cust_rst_outg	ctrl_zone_key
event_details.re_key	cf_cust_rst_outg	cause_ctrl_zone_key
service_point_supply_nodes.cu_duration	cf_cust_rst_outg	outg_duration, cmi
event_details.e_num_momentaries	cf_cust_rst_outg	num_momentary
event_supply_nodes.dv_key	cf_cust_rst_outg	aff_device_key
event_details.dv_key	cf_cust_rst_outg	cause_device_key

EVENT_CREWS

The data in the EVENT_CREWS table is migrated to the CF_RST_CREW table. The primary key will be populated from the SPL_RST_CREW_SEQ.NEXTVAL sequence that is normally used by the BI load process.

EVENT_CREWS Field	BI Table Name	BI Field Name
cr_key	cf_rst_crew	crew_key
e_key	cf_rst_crew	event_key
ecr_crew_assn_time (assign_date_key)	cd_date	cal_dt
ecr_crew_assn_time (assign_time_key)	cd_time	src_time
ecr_crew_uassn_time (unassign_date_key)	cd_date	cal_dt
ecr_crew_uassn_time (unassign_time_key)	cd_time	src_time
ecr_crew_acpt_time (accept_date_key)	cd_date	cal_dt
ecr_crew_acpt_time (accept_time_key)	cd_time	src_time
ecr_crew_arrv_time (arrive_date_key)	cd_date	cal_dt
ecr_crew_arrv_time (arrive_time_key)	cd_time	src_time

EVENT_CREWS Field	BI Table Name	BI Field Name
ecr_crew_cmpl_time (cmpl_date_key)	cd_date	cal_dt
ecr_crew_cmpl_time (cmpl_time_key)	cd_time	src_time
ecr_crew_assn_user (assign_user_key)	cd_user	user_cd
ecr_crew_uassn_user (unassign_user_key)	cd_user	user_cd
ecr_crew_acpt_user (accept_user_key)	cd_user	user_cd
ecr_crew_arrv_user (arrive_user_key)	cd_user	user_cd
ecr_crew_cmpl_user (cmpl_user_key)	cd_user	user_cd
ecr_crew_work_dur	cf_rst_crew	WORK_ DURATION
ecr_crew_assn_dur	cf_rst_crew	ASSIGN_ DURATION
ecr_crew_disp_dur	cf_rst_crew	DISPATCH_ DURATION
ecr_crew_inroute_dur	cf_rst_crew	INROUTE_ DURATION

INDICE

The INIDICE table in Performance Mart is not migrated in the normal migration script. This is because the Indice calculations can be performed for a specific month by running this SQL*Plus command, replacing the 31-JAN-2004 with a month to calculate indice data for:

```
declare temp NUMBER;
begin
    temp := SPL_OMS_SNAPSHOT_PKG.spl_ctrl_zone_outg_snap_fnc( FALSE,
'M', to_date( '31-JAN-2004', 'DD-MON-YYYY' ), 4, 1, NULL, 3, 5, 'NORM'
);
    commit;
    temp := SPL_OMS_SNAPSHOT_PKG.spl_city_outg_snap_fnc( FALSE, 'M',
to_date( '31-JAN-2004', 'DD-MON-YYYY' ), 4, 1, NULL, 3, 5, 'NORM' );
    commit;
end;
/
```

The INDICE data is now stored in two BI tables: CF_CTRL_ZONE_OUTG and CF_CITY_OUTG. The records in the INDICE table that have an RE_KEY with a 'CIR' type will be stored in the CF_CTRL_ZONE_OUTG table, and those with a 'CITY' type will be stored in the CF_CITY_OUTG table.

These two tables also store the data that was stored in the REPORTING_ELEMENT_FACTS table for customer counts.

The following table defines the BI CF_CTRL_ZONE_OUTG table, and describes if possible where the corresponding data use to exist in Performance Mart. The fields in the CF_CITY_OUTG table have similar descriptions, so they will not be described here.

BI Field Name	Description	Corresponding Performance Mart Field
CTRL_ZONE_KEY	Foreign Key to the Control Zone Table.	INDICE.RE_KEY
TMED_IND	Does this calculation include data that was excluded due to occurring during a Major Event	INDICE..TMED_EXCLUDED
SNAP_TYPE_CD	Snapshot Type (M – Month, Y – Year, …)	N/A
SNAPSHOT_DATE_KEY	Date that the Indice data was calculated	INDICE.INDICE_DATE
BEGIN_DATE_KEY	Begin Date of the Period for which Indice calculations were performed	N/A
END_DATE_KEY	End Date of the Period for which Indice calculations were performed	N/A
NUM_CUST_SERVED	Average Number of Customers that were present in the Region during the Period	REPORTING_ELEMENTS_ FACTS. REF_CUSTOMERS_SERVED
NUM_SUST_INTRPT	Total Number of Sustained Interruptions during the snapshot period	SUM(INDICE. INTERRUPTIONS) where DURATION > 5
NUM_MOM_INTRPT	Total Number of Momentary Interruptions during the snapshot period	SUM(INDICE. INTERRUPTIONS) where DURATION < 5
CMI	Total Customer Minutes Interrupted during the snapshot period	SUM(INDICE. INTERRUPTIONS * INDICE.DURATION)

BI Field Name	Description	Corresponding Performance Mart Field
NUM_MULT_SUST_INTRPT	Total number of Customers that Experienced more than a certain number of Sustained interruptions during the snapshot period.	Calculated when a CEMI report is run.
NUM_MULT_CUST_INTRPT	Total number of Customers that Experienced more than a certain number of sustained or momentary interruptions during the snapshot period.	Calculated when a CEMSMI report is run.
SAIDI	SAIDI	Calculated when a SAIDI report is run.
CAIDI	CAIDI	Calculated when a CAIDI report is run.
SAIFI	SAIFI	Calculated when a SAIFI report is run.
CEMI	CEMI	Calculated when a CEMI report is run.
CEMSMI	CEMSMI	Calculated when a CEMSMI report is run.
CAIFI	CAIFI	Calculated when a CAIFI report is run.
MAIFI	MAIFI	Calculated when a MAIFI report is run.
MAIFIE	MAIFIe	Calculated when a MAIFIE report is run.
ASAI	ASAI	Calculated when a ASAI report is run.
ACI	ACI	Calculated when a ACI report is run.
MSAIFI	MSAIFI	Calculated when a MSAIFI report is run.
NUM_EVENT	Number of Distinct Events in NMS during the snapshot period	COUNT(DISTINCT INDICE.EVENT_KEY)

BI Field Name	Description	Corresponding Performance Mart Field
NUM_CUST_INTRPT	Total number of Customers that experienced one or more interruptions during the period	COUNT(DISTINCT INDICE.CUSTOMER)
NUM_MOM_E_INTRPT	Total number of Momentary Events that proceeded a lockout	SUM(INDICE.MAIFIE_INTERRUPTIONS)

NRT Table Mapping

The NRT data will not be migrated from the Performance Mart database, as this is transitional data and will need to be populated from the NMS database once a system is upgraded to support the BI extraction process.

However, the following table mappings are here to help with report conversion projects, and will map how the data would have been migrated if the Performance Mart NRT tables were migrated. Most the data from the NRT tables will be mapped to CF*RECENT* tables, with the exception that some textual data will be stored in either the CD_EVENT or CD_CALL_INFO tables, as described in the following sections.

Also, if a field is not listed in a mapping, then the data is not extracted from the Network Management System database to the BI database with the default product extractors. If missing data is required, then a project configuration change to the NMS extractors will have to be made to get the data into one of the UDF/UDM fields available in BI.

NRT_EVENT_CALL_FACTS

The data in the NRT_EVENT_CALL_FACTS table exists in two different BI tables, one dimension and one fact: CD_CALL_INFO and CF_RECENT_CALL. The following table shows which fields go into which table. For the BI tables below that are not CD_CALL_INFO or CF_RST_CALL, the mapping is done by using the foreign key in the CF_RECENT_CALL table. For example, to get the NRT_ECF_ACCOUNT_NUMBER, the CF_RECENT_CALL table would be joined to the CD_ACCT table by ACCT_KEY.

NRT_EVENT_CALL_FACTS Field	BI Table Name	BI Field Name
nrt_ecf_incident_number	cd_call_info	src_incident_id
nrt_ecf_last_name and nrt_ecf_first_name	cd_call_info	caller_name
nrt_ecf_area_cod and nrt_ecf_phone_number and nrt_ecf_phone_extension	cd_call_info	phone_nbr
nrt_ecf_complaint	cd_call_info	Complaint
nrt_ecf_operator_comment	cd_call_info	Comments
nrt_ech_short_desc	cd_call_info	udf3_descr
nrt_active	cd_call_info	udf1_cd

NRT_EVENT_CALL_FACTS Field	BI Table Name	BI Field Name
nrt_ecf_incident_number	cf_recent_call	src_incident_id
nrt_ecf_account_number	cd_acct	src_acct_id
nrt_ecf_total_priority	cf_recent_call	priority_ind
ecf_called_time (Date)	cd_date	cal_dt
ecf_called_time (Time)	cd_time	src_time
nrt_user_name	cd_user	user_cd

NRT_EVENT_DETAILS

The data in the NRT_EVENT_DETAILS table is available in two different BI tables, one dimension and one fact: CD_EVENT and CF_RECENT_JOB. The following table shows which fields go into which table.

EVENT_DETAILS Field	BI Table Name	BI Field Name
nrt_outage_number	cd_event	src_nbr
nrt_event_idx	cd_event	event_nbr
nrt_ops_exclude_reason	cd_event	exclude_reason
nrt_operator_comment	cd_event	operator_comment
nrt_valid_state_key	cd_event	event_state_descr
nrt_event_status	cd_event	event_state_cd
nrt_street_address || ',' || nrt_city_state	cd_event	first_call_addr
nrt_trouble_code	cd_event	trouble_cd_list
X_coord	cd_event	X_coordinate
Y_coord	cd_event	Y_coordinate
nrt_outage_number	cf_recent_job	src_job_nbr
nrt_status + 1	cf_recent_job	event_status_key
nrt_begin_time	cf_recent_job	begin_dttm
nrt_completion_time	cf_recent_job	rst_dttm
nrt_est_restore_time (est_rst_date_key)	cd_date	cal_dt
nrt_est_restore_time (est_rst_time_key)	cd_time	src_time
nrt_ops_exclude_flag	cf_recent_job	oms_exclude_ind
nrt_cancel_flag	cf_recent_job	cancelled_ind
re_key	cf_recent_job	ctrl_zone_key

EVENT_DETAILS Field	BI Table Name	BI Field Name
dv_key	cf_recent_job	device_key
nrt_ops_cust	cf_recent_job	udm1

NRT Customer Outage

Customer Outage information is stored in three key NRT tables in Performance Mart: NRT_SERVICE_POINT_SUPPLY_NODES, NRT_EVENT_SUPPLY_NODES and NRT_EVENT_DETAILS. Data from each of these tables as well as Customer Keys in the CUSTOMER_SERVICE_POINTS table will be available in the CF_CUST_RECENT_OUTG table in BI.

NRT Fields	BI Table Name	BI Field Name
customer_service_points.cu_serv_loc_key	cf_cust_nrt_outg	acct_key, prem_key, addr_key
customer_service_points.cu_cust_key	cf_cust_nrt_outg	per_key
customer_service_points.cu_meter_key	cf_cust_nrt_outg	meter_key
nt_event_supply_nodes.nrt_outage_time	cf_cust_recent_outg	begin_dttm
nrt_eventsupply_nodes.when_restored_time	cf_cust_recent_outg	rst_dttm
nrt_event_supply_nodes.re_key	cf_cust_recent_outg	ctrl_zone_key
nrt_event_details.re_key	cf_cust_recent_outg	cause_ctrl_zone_key
nrt_event_supply_nodes.nrt_esn_duration	cf_cust_recent_outg	outg_duration
nrt_event_supply_nodes.dv_key	cf_cust_recent_outg	aff_device_key
nrt_event_details.dv_key	cf_cust_recent_outg	cause_device_key
nrt_event_supply_nodes.level1_name	cd_ctrl_zone	udf1_descr
nrt_event_supply_nodes.level2_name	cd_ctrl_zone	udf2_descr
nrt_event_supply_nodes.level3_name	cd_ctrl_zone	udf3_descr
nrt_event_supply_nodes.level4_name	cd_ctrl_zone	udf4_descr
nrt_event_supply_nodes.level5_name	cd_ctrl_zone	udf5_descr
nrt_event_supply_nodes.level6_name	cd_ctrl_zone	udf6_descr
nrt_event_supply_nodes.num_crit_c_cust_out	cd_prem	count(*) where udf6_cd = 1
nrt_event_supply_nodes.num_crit_d_cust_out	cd_prem	count(*) where udf7_cd = 1
nrt_event_supply_nodes.num_crit_k_cust_out	cd_prem	count(*) where udf8_cd = 1

NRT_EVENT_CREWS

The data in the NRT_EVENT_CREWS table is available in the CF_RECENT_CREW table.

EVENT_CREWS Field	BI Table Name	BI Field Name
nrt_ecr_crew_assn_time (assign_date_key)	cd_date	cal_dt
nrt_ecr_crew_assn_time (assign_time_key)	cd_time	src_time
nrt_ecr_crew_uassn_time (unassign_date_key)	cd_date	cal_dt
nrt_ecr_crew_uassn_time (unassign_time_key)	cd_time	src_time
nrt_ecr_crew_acpt_time (accept_date_key)	cd_date	cal_dt
nrt_ecr_crew_acpt_time (accept_time_key)	cd_time	src_time
nrt_ecr_crew_arrv_time (arrive_date_key)	cd_date	cal_dt
nrt_ecr_crew_arrv_time (arrive_time_key)	cd_time	src_time
nrt_ecr_crew_cmpl_time (cmpl_date_key)	cd_date	cal_dt
nrt_ecr_crew_cmpl_time (cmpl_time_key)	cd_time	src_time
nrt_ecr_crew_assn_user (assign_user_key)	cd_user	user_cd
nrt_ecr_crew_uassn_user (unassign_user_key)	cd_user	user_cd
nrt_ecr_crew_acpt_user (accept_user_key)	cd_user	user_cd
nrt_ecr_crew_arrv_user (arrive_user_key)	cd_user	user_cd
nrt_ecr_crew_cmpl_user (cmpl_user_key)	cd_user	user_cd
nrt_ecr_crew_work_dur	cf_recent_crew	WORK_ DURATION
nrt_ecr_crew_assn_dur	cf_recent_crew	ASSIGN_ DURATION
nrt_ecr_crew_disp_dur	cf_recent_crew	DISPATCH_ DURATION
nrt_ecr_crew_inroute_dur	cf_recent_crew	INROUTE_ DURATION

Migration Requirements

Before running the migration script, make sure that:

- The current Performance Mart and NMS databases must be accessible to the BI database using database links that will be created in the BI DWADM database account.
- The BI database must be installed following the installation instructions in the *Oracle Business Intelligence Installation Guide.*
- The following Unix environment variables point to the Performance Mart and NMS database.

  CES_DM_USER - Oracle Username for the Performance Mart Database

  CES_DM_PASSWD - Password for the CES_DM_USER user

  CES_DM_INSTANCE - SQL*Net connection to the Performance Mart Database

  RDBMS_HOST - SQL*Net connection to the NMS database

- The following two environment variables can be set if the default settings create errors when the migration script is run.
  - **CES_DM_DBLINK** - Name of the Database Link created in the BI Oracle account to point to the Performance Mart Database. If this is not set, then the value in the CES_DM_INSTANCE environment variable is used.
  - **CES_OPS_DBLINK** - Name of the Database Link created in the BI Oracle account to point to the NMS database. If this is not set, then the value in the RDBMS_HOST environment variable is used.
- Verify that you have adequate storage. The storage requirements for the BI database will be similar to the current storage requirements for the Performance Mart database. So if the data in Performance Mart takes up 5 GB of space, then a good estimate for BI storage requirement will be 5 GB.
- The following additional Unix environment variables must be set:
  - **CES_BI_USER** - Oracle Username that owns the BI data tables. Normally this will be DWADM.
  - **CES_BI_PASSWD** - Password for the CES_BI_USER user.
  - **CES_BI_INSTANCE** - SQL*Net connection to the BI Database.
  - **CES_BI_DATA_SOURCE** - Data Source Indicator that will be used when storing the migrated records in the BI tables. This should match the value in the AP_MIN_VALUE field in the APPLICATION_PARAMS table where the AP_NAME = 'DATA_SOURCE_INDICATOR'. The default setting of this is 4.
  - **CES_SQL_FILES** - Directory name where the NMS SQL files are stored. Normally this will be $HOME/sql. This is used by the migration script to find the project sql files.

Running the Migration Script

The migration script, **migrate_business_intelligence,** will exist in the $HOME/bin directory of the NMS Unix account. It can be run from this directory, as long as the requirements mentioned in the preceding section are complete.

The migration script takes no parameters, and can be run from the bin directory using this command.

```
nohup ./migrate_business_intelligence > migrate_business_intelligence.out &
```

This will create two log files. The migrate_business_intelligence.out log file can be monitored while the script is running, and the migrate_business_intelligence.log file will be updated once the migration script is completed.

For project-specific migration issues, the following two files will be called from the migration script: project_migrate_bi_dim.sql and project_migrate_bi_fact.sql. The project_migrate_bi_dim.sql will be called after all of the dimension tables are populated by the product migration script, but before the fact tables are populated, so that records will exist in all of the dimension tables for foreign keys in the fact tables. Then the project_migrate_bi_fact.sql will be called after the fact tables are populated, but before the BI Sequences are reset. If either of these two files don't exist in the sql directory, the following messages may appear in the output file:

```
SP2-0310: unable to open file "project_migrate_bi_dim.sql"
SP2-0310: unable to open file "project_migrate_bi_fact.sql"
```

If either of these two messages appear, and the corresponding project migration script has not been created, then these errors can be ignored.

Once the migration completes, there should be data in the following BI tables, matching the records that exist in Performance Mart.

- cd_acct
- cd_addr
- cd_call_info
- cd_city
- cd_crew
- cd_ctrl_zone
- cd_device
- cd_event
- cd_event_status
- cd_meter
- cd_per
- cd_prem
- cd_snl
- cd_user
- cf_cust_rst_outg
- cf_rst_job
- cf_rst_call
- cf_rst_crew

If data is migrated from Performance Mart to BI, then the datafiles generated by the initial extractor runs of all the extractors must not be loaded into BI. Otherwise, all of the active records already stored in BI will be marked inactive, and new records generated, causing a large increase in record counts in the BI tables with no benefit. For this reason, the NMS must be shutdown while the migration is run and the new BI extractors must be run once. Otherwise, the potential exists for losing data that changed after the migration was run but before the new BI extractors are initially run.

To work around this issue, the LAST_START_DATE and LAST_COMPLETE_DATE in the BI_EXTRACTOR_LOG table in the NMS database can be updated with this command once the last Performance Mart extract is run.

```
UPDATE bi_extractor_log

SET last_start_date = SYSDATE, last_complete_date = SYSDATE

WHERE extractor_name NOT LIKE 'NRT%';
```

Note that to do this update, the NMS database must have been migrated and the install_business_intelligence script run to create the BI extractor code.

Troubleshooting Migration Issues

The following sections describe some common troubleshooting scenarios and the resolution.

Cannot Delete from CD_USER table

If the BI Demo environment was installed, then existing records in the CC&B fact tables can point to existing records in the CD_USER table, which will keep the delete of the CD_USER records from running. The migration script deletes all of the OMS data, but does not modify any existing CC&B or EAM records. So if you need to delete the CC&B data in order to delete the demo records in the CD_USER table, the following deletes must be done in the BI database prior to running the migration script:

```
delete from CF_FT;
delete from CF_CASE;
delete from CF_CASE_LOG;
delete from CF_CC;
```

This will not delete all of the CC&B demo data, but will delete the records that refer to CD_USER records that the migration script needs to delete.

No Data in the CF_RECENT* tables

As mentioned in the NRT Table Mapping section above, the NRT data is not migrated during the migration run. This data will be populated by extracting the NRT data from the NMS database and loading it into the BI Database.

No Data in the CF_CTRL_ZONE_OUTG, CF_CITY_OUTG or CF_OUTG tables

The CF_CTRL_ZONE_OUTG and CF_CITY_OUTG tables are a replacement for the INDICE table in Performance Mart. However, the data in these tables can be calculated based on the records in the CF_CUST_RST_OUTG tables, so migration of this data was not done. If records are required for these tables in the BI database, then the SPL_OMS_SNAPSHOT_PKG.SPL_CTRL_ZONE_OUTG_SNAP_FNC or the SPL_OMS_SNAPSHOT_PKG.SPL_CITY_OUTG_SNAP_FNC can be run for the periods that data is required for.

The CF_OUTG table is a snapshot table, that must be refreshed every hour by running the SPL_OMS_SNAPSHOT_PKG.SPL_OUTG_SNAP_FNC function from OWB. As this data is not available in Performance Mart, no migration was possible. This data will need to be captured from the running BI database as it is used.

Snapshots

This section presents an example call to populate snapshot tables CF_CTRL_ZONE_OUTG and CF_CITY_OUTG for last month. This really only needs to be run once a month, sometime after the last changes are made to data in NMS for the previous month and extracted to BI.

Control Zone Outage Snapshot

```
declare temp NUMBER;
begin
    temp := SPL_OMS_SNAPSHOT_PKG.spl_ctrl_zone_outg_snap_fnc( FALSE,
'M', ADD_MONTHS( LAST_DAY( SYSDATE ), -1 ),
                       4, 1, NULL, 3, 5, 'NORM' );
    commit;
end;
/
```

City Outage Snapshot

```
declare temp NUMBER;
begin
    temp := SPL_OMS_SNAPSHOT_PKG.spl_city_outg_snap_fnc( FALSE, 'M',
ADD_MONTHS( LAST_DAY( SYSDATE ), -1 ),
                       4, 1, NULL, 3, 5, 'NORM' );
    commit;
end;
/
```

To create a Daily Indice record set, you would change the P_SNAP_TYPE_CD, which is now 'M' for Monthly, to 'D' for Daily, and also change ADD_MONTHS(LAST_DAY(SYSDATE), -1) to TRUNC(SYSDATE - 1) to create statistics for yesterday.

The CF_OUTG table is populated from a Workflow that you can schedule to run. It takes information from the CF*RECENT tables, and calculates an hourly snapshot, so this can be scheduled to run after the RECENT records have been loaded once an hour.

For more information on Snapshots and their parameters, please see the Oracle Business Intelligence Help. To display the online help, press the button () located in the Business Intelligence Action Bar at the top of any portal screen.

Chapter 12

User Authentication

This chapter describes how to configure authentication of users for the Oracle Utilities Network Management System (NMS) applications.

- Overview of Authentication
- Configuring the WebLogic Security Realm
- Configuring Authentication Using WebLogic Internal Users/Groups
- Configuring Authentication Using an ActiveDirectory Provider
- Configuring Authentication Using an OpenLDAP Provider

Overview of Authentication

To use NMS, a user has to be configured for both authentication and authorization.

Authentication (*i.e.*, user names and passwords) for Oracle Utilities Network Management System is handled by WebLogic, and is accomplished by configuring authentication providers in WebLogic's default security realm. This is a simplification from previous releases, where user names and passwords were kept in database tables, or where LDAP or Active Directory information had to be configured in SQL files.

Authorization (*i.e.*, what applications a user is allowed to use, with what role or user type, or whether the user is allowed to login to the NMS at all) is handled by the Configuration Assistant. See chapter 16 of the Oracle Utilities Network Management System User's Guide for more information on the use of the Configuration Assistant.

Most installations will want to configure WebLogic to use an external authentication source, such as Active Directory or LDAP. These servers are often readily available on most corporate networks, they provide advantages for enforcing security policies (e.g., password complexity and aging), and the login names and passwords are already familiar to the end users. In the case that a more simple solution is required, WebLogic internal users and groups can be used to authenticate against the NMS, although this is not recommended for production environments.

Any user that appears in the users and groups in WebLogic's default security realm tab can be configured to login to NMS, with the following two conditions:

- The user must exist in the WebLogic group **nmsuser** (the name of this group can be changed in $NMS_CONFIG/jconfig/build.properties, if necessary).
- The user must be added to **NMS** through the Configuration Assistant. This will add the user to the CES_USER and USER_PERMISSIONS tables.

Without both of these conditions being met, the application will return that the user is unauthorized.

Configuring the WebLogic Security Realm

1. Login to the WebLogic Administration Console
2. In the Domain Structure pane, click on Security Realms.
3. Click on the default security realm (typically called myrealm).
4. Click on the Providers tab.
5. Click on DefaultAuthenticator.
6. Change Control Flag so it is set to OPTIONAL.

Configuring Authentication Using WebLogic Internal Users/ Groups

The following steps can be used to create users and groups directly in the WebLogic default security realm.

1. Login to the WebLogic Administration Console
2. In the Domain Structure pane, click on Security Realms
3. Click on the default security realm (typically called myrealm).
4. Click on the Users and Groups tab, and then click on the Groups tab.
5. Click on the New button to create a new group.
6. Enter the following group properties:

   Name: *nmsuser*

   Description: Group membership for NMS login.

   Provider: DefaultAuthenticator
7. For each user to be created, click on the **Users** tab, and press the **New** button to create a new user. Enter the following user properties:

   Name: juser

   Description: Joe User

   Provider: DefaultAuthenticator

   Password: *****

   Confirm Password: *****

   > **Note:** User names must be unique. Passwords must contain at least one special character.
8. For each user created, click on that user name in the list of users. Click the **Groups** tab, select the **nmsuser** group from the list of available groups, and move it to the **Chosen** list by using the **>** button. Click **Save**.

Configuring Authentication Using an Active Directory Provider

This section provides an example for how to connect WebLogic to an Active Directory. The specifics of your Active Directory domain may differ from the example given, so consult with your Active Directory administrator to find the correct values, and refer to the WebLogic documentation for specifics on each option.

1. Login to the WebLogic Administration Console.
2. In the **Domain Structure** pane, click on **Security Realms**.
3. Click on the default security realm (typically called myrealm).
4. Click on the **Providers** tab and click the **New** button.
5. Provide a name for the provider (for example, "nms-provider"), and select **ActiveDirectoryAuthenticator** as the type.
6. Click the name of the newly created provider.
7. Under the **Configuration** tab, select the **Common** tab, and set **Control Flag** to **Optional**.
8. Click **Save**.
9. Under the **Configuration** tab, select the **Provider Specific** tab, and set desired values that match your Active Directory configuration.

   Examples:

   <u>**Connection**</u>

   Host: server.example.com

   Port: 389

   Principal: cn=Administrator,cn=Users,dc=example,dc=com

   Credential: (the password used to connect to the account defined by Principal)

   <u>**Users**</u>

   User Base DN: cn=Users,dc=example,dc=com

   User From Name Filter: (&(samAccountName=%u)(objectclass=user))

   User Name Attribute: samAccountName

   User Object Class: user

   <u>**Groups**</u>

   Group Base DN: cn=Groups,dc=example,dc=com

   Group From Name Filter: (&(cn=%g)(objectclass=group))

10. Click **Save**.
11. In the **Change Center**, click **Activate Changes**.
12. Restart the AdminServer.
13. **IMPORTANT:** Verify that the users and groups from the Active Directory are configured by looking at the Users and Groups tab under the default security realm. If not, adjust the configuration.

Configuring Authentication Using an OpenLDAP Provider

This section provides an example of how to connect WebLogic to an OpenLDAP server. The specifics of your OpenLDAP directory may differ from the example given, so consult with your LDAP administrator to find the correct values, and refer to the WebLogic documentation for specifics on each option.

1. Login to the WebLogic Administration Console.
2. In the **Domain Structure** pane, click on **Security Realms**.
3. Click the default security realm (typically called myrealm).
4. Click the **Providers** tab and press the **New** button.
5. Provide a name for the provider (for example, “nms-provider”), and select **OpenLDAPAuthenticator** as the type.
6. Click the name of the newly created provider.
7. Under the **Configuration** tab, select the **Common** tab, and set **Control Flag** to **Optional**.
8. Click **Save**.
9. Under the **Configuration** tab, select the **Provider Specific** tab, and set desired values that match your LDAP Directory configuration.

Examples:

<u>**Connection**</u>

Host: server.example.com

Port: 389

Principal: cn=Manager,dc=example,dc=com

Credential: (the password used to connect to the account defined by Principal)

<u>**Users**</u>

User Base DN: ou=Users,dc=example,dc=com

User from Name Filter: (&(uid=%u)(objectclass=inetOrgPerson))

User Name Attribute: uid

User Object Class: inetOrgPerson

<u>**Groups**</u>

Group Base DN: ou=groups,dc=example,dc=com

Group From Name Filter: (&(cn=%g)(objectclass=groupOfNames))

10. Click **Save**.
11. In the **Change Center**, click **Activate Changes**.
12. Restart the AdminServer.
13. **IMPORTANT:** Verify that the users and groups from the LDAP server are configured by looking at the Users and Groups tab under the default security realm. If not, adjust the configuration.

Chapter 13

Fault Location, Isolation, and Service Restoration Administration

This chapter describes how to configure and administer Fault Location, Isolation, and Service Restoration (FLISR). It includes the following topics:

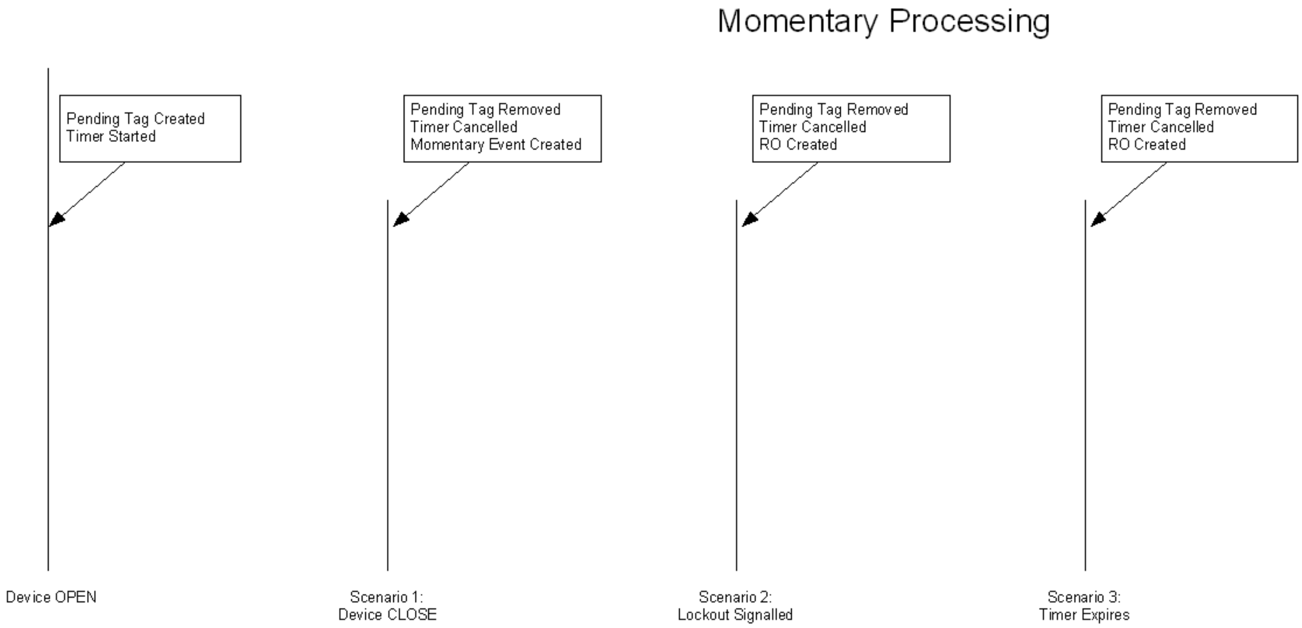
- **Introduction**
- **Fault Location, Isolation, and Service Restoration Timeline**
- **Software Architecture Overview**
- **Configuring Classes and Inheritance**
- **SRS Rules**
- **High Level Messages**
- **Troubleshooting**

Introduction

The intended audience for this document is the system administrators responsible for maintaining the Oracle Utilities Network Management System.

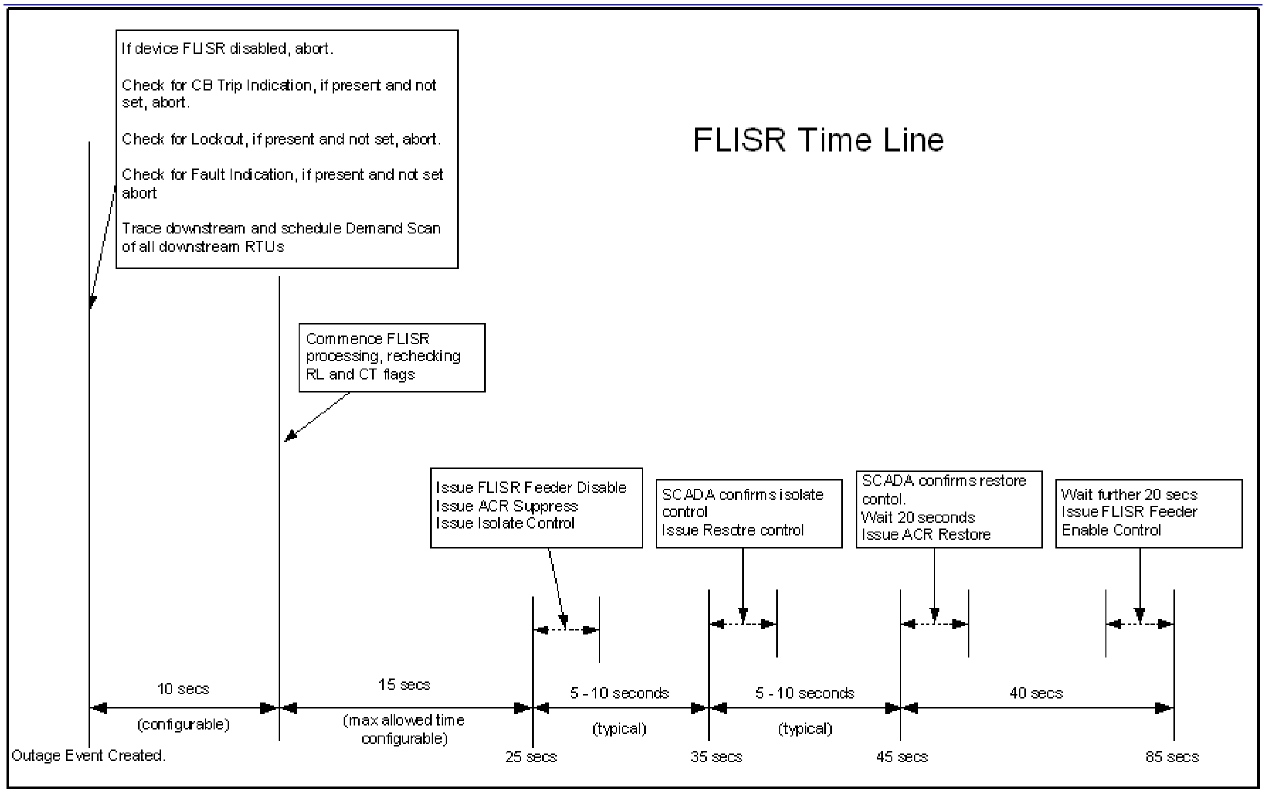
Fault Location, Isolation, and Service Restoration Timeline

These figures show the sequence of events in a Fault Location, Isolation, and Service Restoration scenario. The following figure shows the various scenarios in the momentary processing.



Note: RO is created only if customer supply nodes are de-energized as a result of the operation.

Once an RO is created, the Fault Location, Isolation, and Service Restoration processing sequence shown in the following figure is initiated.



The control sequence (starting at around 25 seconds) is only performed in automatic mode. In manual mode an operator must initiate the control sequence.

Timings in the above diagram are only indicative. Actual values will depend on the complexity of the solution required and the responsiveness of the isolate/restore controls sent to SCADA. The following timings are deterministic:

- The delay allowed for demand scans. This is configurable and defaults to 10 seconds
- The maximum time allowed for the solution in automatic mode. This is configurable and defaults to 15 seconds. If the solution takes longer to solve than this time, Fault Location, Isolation, and Service Restoration will not automatically execute the control sequence. The option for an operator to manually initiate the control sequence is preserved though.
- Maximum time allowed for automatic operations after the lockout is: Demand scan delay + 15 seconds (25 seconds in the default configuration).
- Wait times for Auto-Reclose operations. These are 20 seconds.

Software Architecture Overview

This section describes the role of various software components in implementing the Fault Location, Isolation, and Service Restoration functionality:

Component	Description
DDService	Tracks SCADA measurements, device operations and Conditions. DDService is the starting point for Fault Location, Isolation, and Service Restoration events. When a device trips, a pending operation is created. When the lockout occurs a completed device operation is sent to MTService. If the breaker is able to reclose – only a momentary event is created. DDService is also responsible for executing Fault Location, Isolation, and Service Restoration switch plans, both in manual and automatic mode. In manual mode the request to execute the switch plan can be initiated by the operator from the Switch Sheet Editor tool. In automatic mode the Fault Location, Isolation, and Service Restoration sub-system requests the switch sheet execution by DDService
PFService	The core of Fault Location, Isolation, and Service Restoration functionality. It contains most of the Fault Location, Isolation, and Service Restoration sub-system. Its initial task is to process device operations from DDService and determine the extent of energisation changes in the modelThese changes are also calculated by MTService and propagated to JMService for outage processing. If the device operation is a trip, the Fault Location, Isolation, and Service Restoration sub-system will perform an initial trace to initiate a demand scan of affected RTUs. The bulk of Fault Location, Isolation, and Service Restoration processing is triggered by JMService deciding that event has de-energised customers. In this scenario JMService instructs PFService to initiate Fault Location, Isolation, and Service Restoration processing. PFService then calculates the various isolate and restore scenarios and populates the database tables with the solutions.
JMService	Receives notifications from MTService about changes in energization on the network. JMService will determine if these changes de-energises customers and if so creates an outage event and informs PFService that Fault Location, Isolation, and Service Restoration processing of that event is required.

Component	Description
WorkAgenda	Monitors notifications from JMService about the creation, update and completion of events. WorkAgenda is configured to highlight Fault Location, Isolation, and Service Restoration events in various ways: • Events detected as potential Fault Location, Isolation, and Service Restoration events are highlighted with a yellow background. The background stays yellow until a Fault Location, Isolation, and Service Restoration solution is found or a further determination indicates that the event cannot be considered an FLISR event (e.g., all restoring switches or feeders are Fault Location, Isolation, and Service Restoration disabled) • Events for which a viable Fault Location, Isolation, and Service Restoration solution is found are highlighted with a pink background. • Events for which a Fault Location, Isolation, and Service Restoration solution is found, but the solution includes overloads on restoring feeders, are highlighted with a light blue background.
FLISR	Provides a summary of the Fault Location, Isolation, and Service Restoration solution for an event. If an event is found to have a Fault Location, Isolation, and Service Restoration solution, the operator can examine the details of that solution by using this tool. This tool primarily reads the database tables to determine the solution information calculated by PFService. The operator can also manually write, append and/or overwrite the generated switch plan.
Switching	Once a solution is found for the Fault Location, Isolation, and Service Restoration event, a switch plan can be created to execute the solution. The switch plan can be created (and executed) automatically, or it can be created manually. In either scenario the switch plan can be viewed from the Switch Sheet Editor. In manual mode the operator can request that DDService execute the plan. In both manual and automatic mode the operator can watch the results of DDService performing a switch plan execution.

Configuring Classes and Inheritance

Fault Location, Isolation, and Service Restoration utilizes standard class names to determine various features in the model. Devices in a model can be configured to the Fault Location, Isolation, and Service Restoration classes using class inheritance.

The following table lists the classes supported by Fault Location, Isolation, and Service Restoration:

Class Name	Purpose
flisr_cb	Set of SCADA devices that are protective. These are the SCADA devices that can trip when a fault is detected.
flisr_sectionalizer	Set of devices that are SCADA controllable, but are not protective. These devices: • Might have fault indicators on them in order to give better indication of fault locations on the feeder • Will be considered for isolate and restore devices
flisr_fuse	Set of non-SCADA protective devices. These are considered when determining loads and limiting devices
flisr_load	Set of devices that are loads on the network – typically distribution transformers.
flisr_cogen	Set of devices on the network that provide additional supply.
conductor	Set of conductor classes on the network. These are considered when determining limiting devices.
block_flisr	Condition classes. These define tags and conditions that automatically prohibit Fault Location, Isolation, and Service Restoration operations on a device.

Database Views

In order to determine loads and limiting devices Fault Location, Isolation, and Service Restoration needs to know basic load profile information about all devices. The following database VIEWS are required:

FLISR_TRANSFORMER

h_cls	INTEGER	Class number of device
h_idx	INTEGER	Index number of device
kva_rating	FLOAT	Transformer rating in kVA
partition	INTEGER	Model partition for device

FLISR_CONDUCTOR

h_cls	INTEGER	Class number of device
h_idx	INTEGER	Index number of device
amp_rating	FLOAT	Device's rating in amps
voltage	FLOAT	Device's nominal voltage in kV
partition	INTEGER	Model partition for device

FLISR_SWITCH

h_cls	INTEGER	Class number of device
h_idx	INTEGER	Index number of device
amp_rating	FLOAT	Device's rating in amps
voltage	FLOAT	Device's nominal voltage in kV
partition	INTEGER	Model partition for device
flisr_enabled	CHAR	Whether FLISR is enabled for this switch (Y or N)
fla_enabled	CHAR	Whether Fault Location Analysis is enabled for this switch (Y or N)

SRS Rules

The following SRS Rules configure Fault Location, Isolation, and Service Restoration functionality and options:

Rule Name	Description
allowFlisrAutoMode	Allow the operators to put Fault Location, Isolation, and Service Restoration into auto-mode
autoRecloseMeasurementName	SCADA attribute used to indicate recloser suppression
earthLeakageMeasurementName	SCADA attribute for earth leakage
failedQualityBitmask	The bitmask to apply to quality codes to determine if quality is bad.
faultIndicatorMeasurementName	SCADA attribute for Fault Indicators
flisrDemandScanThreshold	Time to wait for demand scans
flisrDisableMeasurementName	SCADA attribute that indicates Fault Location, Isolation, and Service Restoration should be disabled
flisrKVATolerance	KVA Tolerance when comparing loads against ratings

Rule Name	Description
flisrMode	Start up mode for Fault Location, Isolation, and Service Restoration
flisrSwitchPlanType	Type of switch plans to use for Fault Location, Isolation, and Service Restoration
flisrTemplateArEnable	Template containing Fault Location, Isolation, and Service Restoration Reclose Enable actions
flisrTemplateArSuppress	Template containing FLISR Reclose Suppress actions
flisrTemplateBase	Template for FLISR switch plans
flisrTemplateDisable	Template containing FLISR Disable actions
flisrTemplateEnable	Template containing FLISR Enable actions
flisrTemplateIsolate	Template containing FLISR Isolate actions
flisrTemplateRestore	Template containing FLISR Restore actions
flisrTemplateWait	Template containing FLISR Reclose Wait actions
manualOperationMeasurementName	SCADA attribute that indicates manual operation of a device
maxFlisrSolutionTime	How long we allow for solutions in automatic mode
mvarMeasurementName	SCADA attribute for current MVAR
mwMeasurementName	SCADA attribute for current MW
preTripMvarMeasurementName	SCADA attribute for pre-trip MVAR
preTripMwMeasurementName	SCADA attribute for pre-trip MW
recloseLockoutMeasurementName	SCADA attribute used to show recloser lockouts

High Level Messages

PFService accepts the following High Level messages:

```
Action any.PFService <command> <arguments>
```

Where:

Command	Arguments	Description
debug FLISR	<N>	Sets the debug level: 0 = off 1 = demand scan & timing info 2 = Trace 3 = Detailed Information regarding solution 4 = Full debug

Command	Arguments	Description
flisr kva_tolerance	<N>	Sets the capacity tolerance to allow. Where <N> is the new tolerance in kVA
flisr base_flows		Outputs the base conductor flow information
flisr ties		Outputs the ties (open) point summary
flisr alarms		Forces a check for the Fault Location, Isolation, and Service Restoration disabled device alarms
flisr check	ON/OFF	Toggle Fault Location, Isolation, and Service Restoration check mode on/off
flisr reload		Reload measurement configuration
flisr dump		Write internal data structures into log

Troubleshooting

The following high-level messages can be used to turn timing and demand scan information on/off. This is useful in determining that Fault Location Isolation Service Restoration is scanning the correct RTUs and that timing goals are being achieved.

To turn on the messages:

```
Action any.PFService debug FLISR 1
```

To turn off the messages:

```
Action any.PFService debug FLISR 0
```

Chapter 14

Distribution Management Application Configuration

This chapter provides an overview of the configuration and maintenance of Oracle Utilities Distribution Management System applications. It includes the following topics:

- **Environment Settings**
- **Configuring Oracle Utilities Network Management Services**
- **Power Flow Rules**

For DMS installation instructions, see the Oracle Utilities Network Management System Installation Guide.

Environment Settings

This section describes how the Oracle Utilities Network Management System Distribution Management services are configured. These settings should be configured for the applications listed below.

- Feeder Load Management
- Fault Location Isolation & Service Restoration (FLISR)
- Fault Location Analysis
- Power Flow Extensions
- Suggested Switching
- Volt/VAr Optimization
- Web Switching

Configuring Oracle Utilities Network Management Services

PFService – Power Flow Service

The main application that runs the majority of the Oracle Utilities Network Management System Distribution Management business logic is the Power Flow service. If your environment will be running any applications listed in the previous section (except Web Switching and FLISR), you must add the Power Flow Service as a system service by updating the $NMS_HOME/etc/system.dat file. There are 3 main sections where this service needs to be defined: the service, program and instance sections. See the $CES_HOME/templates/system.dat.template file for examples of how to configure the Powerflow Service. Search for PFService in the file and copy those lines to $NMS_HOME/etc/system.dat file. Make sure all lines are uncommented so that they are active. You must restart the system services in order for the Powerflow Service to be properly monitored by SMService.

The command line options for PFService are:

- **hourlyProfiles**: PFService should be run with this option to activate the load interval data functionality
- **incrSolveCutoff**: similar to the MTService -incrSolveCutoff. Default value is 1000 switches. The PFService and MTService parameters should be tuned separately, since PFService performs more actions as part of the solve.
- **pfdbs**: Use a dedicated database connection, rather than the common pool. Requires a corresponding PFDBService instance to be defined in system.dat

Non-Converged Islands

When PFService encounters apparent model errors that preclude a solution for an island, the island is marked as "Non-Converged" and the Power Flow solution attempt is stopped. The island at this point is 'disabled.'

To output the list of disabled islands to the PFService log file:

```
Action any.PFService dump_disabled_islands
```

When the model has been rebuilt with data to solve the error, you may re-enable the island:

```
Action any.PFService reenable_island <source alias or handle>
```

Power Flow Rules

Oracle Utilities Network Management System Distribution Management applications use srs_rules parameters with a SET_NAME of 'PFS' to configure what kind of data sets are used and how the application results are computed and displayed.

To view and edit Power Flow Rules, use the Event Management Rules tab in the Configuration Assistant. Expand the **Power Flow Related Rule** item in the left panel to display the rule categories.

Chapter 15

Java Application Configuration

This chapter describes how to configure and deploy customizations to the Oracle Utilities Network Management System (NMS) Java applications. This chapter includes the following topics:

- **Java Application Configuration Overview**
- **JBot GUI Configuration**
- **Customizing Applications**
- **JBotCommand Methods Reference**

Java Application Configuration Overview

The Oracle Utilities Network Management System Java applications are configured by using the standard product configuration with overrides that are specific to a customer. This chapter describes where the Java application configuration files reside as well as how to update and deploy changes to these files to an Oracle Utilities Network Management System Web Gateway.

This section includes the following topics:

- **Making Changes to Java Application Configuration**
- **Deploying Configuration Changes**
- **Build Process for XML and Properties Files**
- **Testing the Java Client Configuration**

Making Changes to Java Application Configuration

After executing the installation procedures outlined in the Oracle Utilities Network Management System Installation Guide, the product configuration files for all Java applications will be stored in ${CES_HOME}/dist/baseconfig/product/. To make a change to any Java configuration file, you will need to copy the file to ${NMS_CONFIG}/jconfig, using the same directory structure as it exists in the product directory.

For example, to change the AMRInterface.properties file, copy it from ${CES_HOME}/dist/baseconfig/product/server/ to ${NMS_CONFIG}/jconfig/server. Make the customer specific changes on the copied version. Do not change the product version.

Configuration Files

While there is some GUI configuration information stored in the database (e.g., certain menu options), the NMS Java application GUI configuration is primarily contained in XML files and text files. The primary configuration file types are:

- **`*.xml`**: NMS Java application XML configuration files follow the JBot XML schema (jbot.xsd). They are the primary configuration file and must be modified as a whole file to be valid. Most of the attributes in the XML file are either required or have a default value.
- **`*.inc`**: XML snippets that are referenced in the XML files.
- **`*.properties`**: standard Java configuration text files. NMS properties configuration follow a base-plus-delta hierarchy so you only need to include a project version of a properties file when you wish to modify a property and then only need to include the lines that are being modified.

Deploying Configuration Changes

These steps are required after changes have been made to a customer's Java application configuration after the initial installation of the Oracle Utilities Network Management System.

The ${NMS_CONFIG}/jconfig/build.properties file contains various properties that control the configuration build process. The following is a list of the commonly modified values:

project.name	The name of the project/customer. This is displayed in the Help About dialog of any Java GUI applications to identify the application as being configured for this particular customer.
project.tag	This is a CVS tag or other identifier used to identify a particular build of the customer-specific configuration. This is also displayed on the Help About dialog of any Java GUI applications to identify a customer-specific configuration deployment.
dir.localization	If the configuration is based off of a localized (non-English language) version, enter the directory of the localization configuration. Otherwise leave this commented out.
dir.config.deploy	This is the directory where runtime configuration jar files will be created. The default is a staging area ($NMS_HOME/java/deploy), but it is also possible to configure these runtime files to be deployed directly to the application server. Uncomment and update the WebLogic sections if this is desired.

After making customer-specific changes to the java application configuration files and setting up the build.properties file for your environment, create the runtime configuration jar files by running the following command:

```
nms-install-config --java
```

This will create the cesejb.ear file. If the cesejb.ear file is to be deployed to a staging area, they will need to be copied to the appropriate directory for the java application server (*i.e.,* WebLogic) to deploy them.

In addition, this command will create nms-amr.ear and nms-mwm.ear files. These files contain Oracle Utilities Network Management System MultiSpeak Adapter and Oracle Utilities NMS-MWM Adapter, respectively.

Deploying to WebLogic Application Server

To deploy the Oracle Utilities Network Management System application in your domain, follow these steps:

1. Access the WebLogic Server Administration Console by entering the following URL:

   `http://hostname:port/console`

   Here hostname represents the DNS name or IP address of the Administration Server, and port represents the number of the port on which the Administration Server is listening for requests (port 7001 by default).

2. If you have not already done so, in the Change Center of the Administration Console, click **Lock & Edit**.
3. In the left pane of the Administration Console, select **Deployments**.
4. If a previous release of Oracle Utilities Network Management System (cesejb) is in the table:
   - Select the checkbox to the far left of the deployed cesejb application and click **Stop** and choose **Force Stop Now** to stop the application.
   - Select the checkbox to the far left of the deployed cesejb application. Click the **Delete** button at the top or bottom of the Deployments table to delete the cesejb application, then click Yes to confirm your decision.
5. In the right pane, click **Install**.
6. In the Install Application Assistant, locate the cesejb.ear to install. This will be in the directory listed in your build.properties setting "dir.config.deploy".
7. Click **Next**.
8. Specify that you want to target the installation as an application.
9. Click **Next**.
10. Select the servers and/or cluster to which you want to deploy the application.

    Note: If you have not created additional Managed Servers or clusters, you will not see this assistant page.

11. Click **Next**.
12. Click **Next**.
13. Review the configuration settings you have specified, and click **Finish** to complete the installation.
14. To activate these changes, in the Change Center of the Administration Console, click **Activate Changes**.

Build Process for XML and Properties Files

The build process will copy and/or merge all of the .xml and .properties files from the product and project directories to $NMS_HOME/java/working. The build process will then jar up all the files. For XML documents that exist in both the product and project directory, the one in the project directory takes precedence. XML files are not merged during the build process. Properties files however, are handled differently. The build process combines project and product files with the same name into one generated file. Here is an example of how this works:

Project Version of the file:

```
…
BTN_CREW_ICONS.text = Crew Icons
...

Product Version of the file:
…
Workspace.text = Env Mgr
LBL_CONNECTION_STATUS.text = Connection Status
BTN_CREW_ICONS.text = Crew Actions
ONLINE.text = Online
OFFLINE.text = Offline
…
```

Generated Version:

```
# Generated from
projects\jconfig\ops\workspace\properties\Workspace_en_US.properties
# $Id: Workspace_en_US.properties,v 1.3 $
BTN_CREW_ICONS.text = Crew Icons
…
# Generated from
product\jconfig\ops\workspace\properties\Workspace_en_US.properties
Workspace.text = Env Mgr
LBL_CONNECTION_STATUS.text = Connection Status
ONLINE.text = Online
OFFLINE.text = Offline
…
```

Note that the BTN_CREW_ICONS.text that was in the original product file is not merged into the generated file. Therefore the project value is used by the application.

If a project overrides a line, it will be removed from the generated product definitions. If a project only duplicates but does not change the product configuration, then the line is removed from the project configuration in the generated file.

Testing the Java Client Configuration

This section details how to test Java client configuration without deploying the changes to an app server. Changes can be made locally on a client Microsoft Windows machine and immediately tested.

1. Use the **Configuration Assistant** to create the client application installer and install the application you will be testing (e.g., Web Workspace). See the Oracle Utilities Network Management System Installation Guide for complete instructions on creating the installer and installing the client applications.

    Notes:

    - The directions below assume that the client is installed to C:\OracleNMS and the project name is OPAL. The location and the project name can be changed as appropriate.
    - Installer Log Files: The installers create log files that may be used in troubleshooting installation issues. The log files are saved to C:\Documents and Settings\[user]\temp\OracleNMS. The log files names have the following convention: *application_name*.log (*e.g.*, WebWorkspace.log).

2. On the NMS server machine, do the following:

    ```
    cd $NMS_CONFIG
    zip -r $HOME/nms_config.zip jconfig
    cd $NMS_HOME
    zip -r $HOME/java.zip java
    ```

3. Next, transfer them to the client machine.

    `Unzip nms_config.zip` **`to`** `c:\OracleNMS\OPAL`

    `Unzip java.zip` **`to`** `c:\OracleNMS\`

4. Install Apache Ant version 1.8.2. Be sure to put the ant bin directory on the system path. For example, if Apache Ant is installed to C:\apache-ant-1.8.2, add C:\apache-ant-1.8.2\bin to the system path.

5. Create two environment variables (using the Windows control panel):

    - `NMS_CONFIG=c:\OracleNMS\OPAL`
    - `NMS_HOME=c:\OracleNMS`

    You can then modify the configuration in `c:\OracleNMS\OPAL\jconfig`

6. To test the changes do:

    ```
    cd c:\OracleNMS\OPAL\jconfig
    ant clean config
    ```

    or

    ```
    ant config
    ```

    Notes:

    - **ant clean config** will regenerate all of the configuration; you will need to do that when updating to a new release.
    - **ant config** can be used within a session to only update the files that have changed.

7. Finally, run the application as normal. The system will use the local configuration instead of the configuration on the server.

JBot GUI Configuration

JBot in General

Overview

JBot is a system developed by the Oracle Utilities Network Management System group for representing GUI forms as XML documents. Product versions of files are stored in ${CES_HOME}/dist/baseconfig. Project versions are stored under ${NMS_CONFIG}/jconfig.

This document contains a description of the configuration needed for all Oracle Utilities Network Management System Java Tools. This includes configuration to:

- Organize all Java Swing Components visually.
- Attach language-independent text and tooltips to the Components.
- Attach specific logic to user actions on the Components.
- Display specific pieces of data held in memory on the Components.
- Set Components' enabled/editable status dependent upon tool-specific States.

Glossary of Terms

Term	Definition
Command	Specific piece of functionality that is executed when a user acts on the GUI.
Component	A member of or enhancement to the standard Java Swing package, including TextFields, TextAreas, Buttons, Tables, Trees, Panels, etc.
Data Store	Collection of data that may be accessed by any Command and displayed by any Component.
Java	Platform-independent object-oriented computer language.
Properties	Standard Java configuration text file. *.properties files define all text and tooltips for each Component.
Swing	Java library of standard visual Components.
Tag	XML key that describes the Component to be added. Tags look like this: `<tag_name>`.
Tool	A grouping of Oracle Utilities Network Management System -specific functionality that can be used as an Application or an Applet.
Tool State	Tool-specific milestones, set as internal flags, that may be used to configure Components' enabled/editable statuses. (POPULATED, ASSIGNED, and CLEARED are all examples of Tool States.)

JBot Component Gallery

This section contains a sample image of each Component, a description of the Component and the Component's name, which is used in the Component's XML tag.

Component Name/ XML Tag	Description
Button	Single clicking on a button will perform a defined Action.
CheckBox	Allows an item to be marked as selected.
CheckBoxMenuItem	A menu item that has a checkbox next to it when it is selected. It is configured just like a MenuItem.
CollapsiblePanel	Collapsible in the horizontal or vertical direction. The purpose is to save screen real estate. The image and the title are configurable.
ComboBox	A list of elements that defaults to showing one element. To select from all of the elements, click on the arrow. The purpose of this Component is to save screen real estate and to only allow the user a finite set of options.
ControlZoneSelector	Popup display of a Control Zone tree, displaying a specified (default 3) # of levels of the control zone hierarchy to allow user selection of a control zone.
CrewIconsPanel	Specialized panel for Crew Actions window.
DateTimeSelector	Pop up (actually more of a dropdown) calendar that allows the user to specify the date/time. It will follow the specified date/time format set in ces_datefmt.
Label	Text description that is associated with another Component, frequently a TextField.
LabelIndicator	Label whose icon changes with the change in the tool status.
List	Lists can be single or multi-select. The list box will be scrollable when the number of elements exceed the size of the list.
MainPanel, SubPanel	Several Components are placed on a panel to control a section of the GUI.
Menu	Element of a MenuBar that can have MenuItems, RadioButtonMenuItem, CheckBoxMenuItem, or SubMenu, and Separators (horizontal delimiters).
MenuBar	Bar at the top of a panel that contains one or more Menu elements.
MenuItem	Standard text or icon option in a Menu.
PasswordField	A field that works just as a TextField except that it displays asterisks instead of the characters typed.
PopupMenu	Right-click menu with a number of menu items which when selected performs a defined action.
RadioButtonMenuItem	A choice on a menu that is part of a group where only one can be selected at a time. It is configured just like a MenuItem.
RadioGroup	Similar to a CheckBox, but only one item can be selected at a time.

Component Name/ XML Tag	Description
ScrollPane	It provides a scrollable view of a set of Components. When screen real estate is limited, it is used to display a set of Components that is large or whose size can change dynamically.
Slider	A Component that lets the user enter a numeric value bounded by a minimum and maximum value.
StatusBar	Displays messages to the user. It contains a Oracle Utilities Network Management System icon, and can also have a progress bar and text and label indicators.
SplitPane	Split the two panels by a divider that can be dragged in either direction to increase or decrease the size of each panel.
Table	Data is displayed in a tabular format. They can support single or multi-row selection, and cells can display icons and DateTimeSelectors in addition to dates and strings.
TabbedPane	A component that lets the user switch between a group of components by clicking on a tab with a given title and/or icon. Contains one or more Tabs.
TextArea	Allows the user to enter text on multiple lines. When the number of lines exceeds the viewing area, then the Component is scrollable.
TextField	Allows the user to enter text.
TextIndicator	Changes the displayed text when the tool status changes.
ToggleButton	A two-state button that stays in the pressed position the first time it is clicked. The button returns to the unpressed position the second time it is clicked.
ToolBar	Component below a MenuBar on a panel. It can be automatically generated from the MenuBar by setting <ToolBar use_menu="true"/> . Also contains ToolBarItems and Separators.
ToolBarItem	Element of a ToolBar, generally with a specified icon.
ToolContainer	Allows a tool to be contained by another tool.
Tree	Data can be presented in a hierarchical order. If a parent has children, then the parent can be opened to display the children or closed to hide them.
TreeTable	A combination of the tree Component and the table Component. This allows a tree to be displayed with multiple columns. Attributes available: **name**="unique component name" **class**="fully qualified class name that overrides com.splwg.oms.jbot.component.JBotPaneTreeTable" See Tree Table XML for sample configuration.
ViewerPanel	Specialized panel used by the Viewer tool.

JBot XML Schema and XML File

Schemas describes the information required to create a valid XML file, what each element has as its child elements, their attributes, and any restrictions on them. The JBot schema has the following conventions:

- **`Elements`**: every word begins with a capital letter (*e.g.*, **`MainPanel`**, **`SubPanelType`**, etc.).
- **`Attributes`**: every word begins with a lowercase letter; attributes with compound names are separated by underscores (*e.g.*, **`name`**, **`layout_type`**, **`collapse_direction`**, etc.).

Standard JBot Tool XML Configuration

JBot Tool XML files are based on the jbot.xsd schema, which has the following structure:

```
<JbotToolApp>
  <GlobalProperties/>
  <ToolBehavior/>
  <MainPanel>
    <MenuBar/>
    <ToolBar/>
    <PopupMenu/>
    <StatusBar/>
  </MainPanel>
  <BaseProperties>
    <Commands/>
    <Imports/>
    <DataStores/>
    <Dialogs/>
    <Adapters/>
  </BaseProperties>
</JBotToolApp>
```

Element Definitions

- **GlobalProperties**: This section defines properties that are used for tool specific configuration values. All values possible are listed in the product XML files where applicable.
- **ToolBehavior**: Typically defines what commands to run upon opening or closing the dialog.
- **MainPanel**: Defines the GUI layout of the tool. Contains MenuBar, ToolBar, PopupMenu, and StatusBar.
- **BaseProperties**: Contains the configuration that matches JBot names with Java classes.
  - **Commands**: This section defines a command. If a command is used either by the tool or by a dialog called from the tool, it must be listed here.

    It is preferable to refer to **`Commands`** using the (class) **`name`** attribute rather than define the name in a child **`CommandClass`** element.

    For example, the following:

    ```
    <Commands name="CMD_FOO"/>
      ...
    <CommandClass name="CMD_FOO"
     class="com.splwg.oms.client.workagenda.FooCommand"/>
    ```
    is equivalent to:

    ```
    <Command name="com.splwg.oms.client.workagenda.FooCommand"/>
    ```

However, if there is an **`Import`** section, the system will attempt to find the command(s) in each package. Thus, the following:

```
<Command name="com.splwg.oms.client.workagenda.FooCommand"/>
```

becomes:

```
<Command name="FooCommand">
...
<Imports>
  <Import name="com.splwg.oms.client.workagenda"/>
</Imports>
```

- **Imports**: This section defines paths for commands so that a command can be used without specifying the full path.
- **Datastores**: All datastores that are used by the tool or any dialogs called by this tool must be listed here. However, a tool is allowed to use a datastore defined by a different tool, as long as the other tool is loaded first. There are also some instances where a datastore can be defined in the code. This is mainly the case in the crew tools.
- **Dialogs**: All dialogs that can be displayed by this tool must be listed in this section. Also, if any dialogs are displayed from other dialogs defined also must be listed here.
- **Adapters**: This section is no longer necessary. If an existing JBot configuration file has this section, it can be removed without a problem. If such a tag does exist, it is ignored.

Include Elements

Runtime Include Elements - use standard XML based xi:include tags. The included files are delivered to the client and they are combined by the application at runtime. This allows for specific XML code that is repeated to be defined once, but used in multiple places.

To define an include file, xmlns, xmlns:xsi, and xsi:schemaLocation must be defined.

For example given this XML fragment:

```
<Perform name="HLM" category="onMessage"
type="APPLY_SAFETY_FILTERS">
```

should be changed to:

```
<Perform name="HLM" category="onMessage" type="APPLY_SAFETY_FILTERS"
  xmlns="http://www.ces.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ces.com http://localhost/xml/
jbot.xsd">
XML code that will be used by multiple tools
...
</Perform>
```

The include files should be saved with an .xml extension.

To reference this file in another XML file, use the following syntax:

```
<xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
  href="/SafetyStartup.xml" parse="xml"/>
  ...
```

This allows the second XML document to use the XML code defined in the include file. The above example defines filters that will be used by multiple tools within Web Switching. Therefore, when filters need to be changed, they can be changed once and it will be applied to all tools that are using the **`include`** file.

Build Time Include Elements - the main limitation on xi:include tags is that they can only be used to insert a single element. While that approach works fine in the body of a JBot configuration, it doesn't work well for inserting tool properties, actions, or datastores.

In these cases, it is easier to use the build time based <Include> element. In this case the build process that creates the nms_config.jar file will perform the inclusions.

These include files should be saved without any extra attributes, and saved with an .inc extension.

```
<Perform name="HLM" category="onMessage"
type="APPLY_SAFETY_FILTERS">
...
```

To reference the file, use the following syntax:

```
<Include name="SafetyStartup.inc"/>
```

JBot Commands

JBot commands are operations performed as a result of an event. Some examples of events are button presses, table editing and row selecting. Commands are defined in a "Perform" tag. The actual options for the Perform tags vary with the component type.

Here is an example of configuring a command to be run when a menu is selected:

```
<MenuItem name="MNU_EMERGENCY_CONTENT_SELECTION"
 icon="Preferences16.gif">
  <PressPerform>
    <Command value="DisplayDialogCommand">
      <Config name="dialog" value="DLG_EMERGENCY_CONTENT_SELECTION"/>
    </Command>
  </PressPerform>
</MenuItem>
```

This will display the dialog `DLG_EMERGENCY_CONTENT_SELECTION`.

There are many JBot commands available. HTML command documentation is available: `$CES_HOME/documentation/command_doc/index.html.`

All commands accept the following `<Config>` parameters:

- **`runInTool`**: the JBotTool that this command should run in. This defaults to the current tool.
- **`abortable`**: whether this command should be aborted when a previous command aborts. This defaults to *`true`*, but can be configured as *`false`* in the rare case that there is a `Command` that should be executed, say, even when a dialog is canceled that sets the abort flag.

JBot Actions

JBot actions allow you to define a list of commands that can be reused multiple times in a configuration. Defining a command directly on a component works well if there is only one place where the command is needed. However, if there are multiple places where the same commands are called, such as a menu item and a button, this provides a way to only define the action once.

Actions should be defined in the ToolBehavior or DialogBehavior tags.

```
<Action name="ACT_PRINT">
  <Command value="DisplayDialogCommand">
    <Config name="dialog"
    value="DLG_PLANNED_REPORT_CONTENT_SELECTION"/>
  </Command>
  <Command value="GenerateReportCommand" when="GENERATE_REPORT">
    <Config name="report_location" value="/Webswitching/
    PlannedSwitching/PlannedSwitching.xdo"/>
```

```
    <Config name="parameter_datastore"
    value="DS_PLANNED_REPORT_CONTENT"/>
    <Config name="base_file_name" value="SwitchPlan"/>
    <Config name="file_description" value="report"/>
    <Config name="show_progress_dialog" value="true"/>
    <Config name="dest_file_reference" value="REPORT_FILE_REF"/>
    <Config name="dest_datastore" value="DS_PLANNED_REPORT_CONTENT"/>
  </Command>
</Action>
```

Then to use this action, the `ExecuteActionCommand` command should be called:

```
<MenuItem name="MNU_PLANNED_PRINT" icon="Print16.gif"
accelerator="control P">
  <PressPerform>
    <Command value="ExecuteActionCommand">
      <Config name="action" value="ACT_PRINT"/>
    </Command>
  </PressPerform>
</MenuItem>
```

The action is run just like another jbot command. Other commands or actions can also be defined before or after the action command, just like any other jbot command.

Actions can also call other actions, by using the `ExecuteActionCommand` from within another action.

It is also possible for actions to take parameters. See the following example:

```
<Action name="ACT_GGENERATE">
  <Command value="GenerateReportCommand" when="GENERATE_REPORT">
    <Config name="report_location" value="/Webswitching/
    PlannedSwitching/$REPORT_NAME$.xdo"/>
    <Config name="parameter_datastore"
    value="DS_PLANNED_REPORT_CONTENT"/>
    <Config name="base_file_name" value="SwitchPlan"/>
    <Config name="file_description" value="report"/>
    <Config name="show_progress_dialog" value="true"/>
    <Config name="dest_file_reference" value="REPORT_FILE_REF"/>
    <Config name="dest_datastore" value="DS_PLANNED_REPORT_CONTENT"/>
  </Command>
</Action>
...

<MenuItem name="MNU_PLANNED_PRINT" icon="Print16.gif"
accelerator="control P">
  <PressPerform>
    <Command value="ExecuteActionCommand">
      <Config name="action" value="ACT_PRINT"/>
      <Config name="$REPORT_NAME$" value="PlannedSwitching"/>
    </Command>
  </PressPerform>
</MenuItem>
```

This will replace the token **`$REPORT_NAME$`** with the value PlannedSwitching. Any text in the configuration can be replaced this way. You cannot, however, replace the Command names themselves.

If you wish to use an Action defined in a different XML file there are two options.

The first option is if you wish to run the action in the other tool. In that case, you can use the "runInTool" option, like other commands. However, if you wish to run the action in the current tool, even though it is defined in another tool, use the **`tool`** config option.

Validation Toolkit

The validation toolkit provides a way of validating user data, as well as another way of coloring tables or other components. While most JBot commands work off of the underlying data, the validation toolkit works off of GUI widgets.

The **Validation Testing Tool** shows various examples what can be configured using the validation toolkit. The example is in:

`jconfig/ops/test/xml/Validation.xml.`

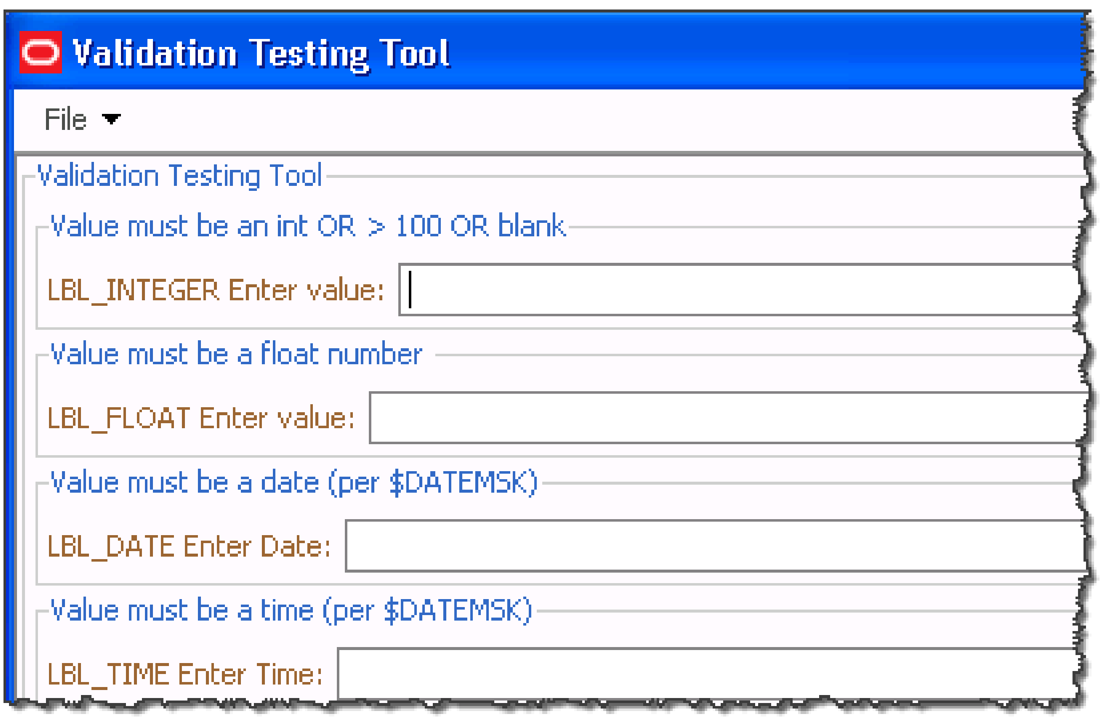
To run this tool, add the following code to `WorkspaceMenuBarTool.xml` to create a menu item that will start the Validation Testing Tool:

```
<MenuItem name="MNUITM_VALIDATION" hide_icon="true">
  <PressPerform>
    <Command value="DisplayToolCommand">
      <Config name="tool" value="Validation"/>
      <Config name="class" value="com.splwg.oms.jbot.JBotTool"/>
    </Command>
  </PressPerform>
</MenuItem>
```

In `WorkspaceMenuBarTool_en_US.properties` add:

```
  MNUITM_VALIDATION.text = Demo Tool...
```

When you click on the new menu choice, it will display the tool:



Feel free to try various values in the form to understand how the validation toolkit can be used. The form is defined by **`Validation.xml`**. To use this form, find the section on the from that is like the validation you wish to perform, then look at `Validation.xml` to see the code needed.

Locale-Specific Properties File

Property Naming Convention

The *<toolname>_<language>_<locale>*.properties file contains all language related properties for the components. They are identified with the syntax:

```
KEY.property = value string
```

Property	Component Type	Description	Syntax	Example
Text	All Actions	String displayed on the Button that invokes the Action.	`ACT_KEY.text = string`	ACT_SET_START_DATE.text = set starting date
text Radio	Button group members, table headers, combo box entries	String displayed on the piece of a larger Component.	`RBG_KEY.ENTRY_KEY.text = string`	RBG_OUTAGE_TABLE.H_IDX.text = Device index
tooltip	All Components and Actions	Tool tip string.	`KEY.tooltip = string`	ACT_SET_START_DATE.tooltip = Set the starting date to the current date

When there is no locale, the tool tries these file names.

1. `<toolname>.properties`
2. `<toolname>_en_US.properties`

If there is a locale defined, the tools will try these file names.

1. `<toolname>_<language>_<locale>.properties`
2. `<toolname>_<language>.properties`
3. `<toolname>.properties`

Reserved Words

Here are some reserved words that you may use in property files. It is recommended that when used, it should be placed at the top of the file.

reserved word	What it does
include	List of additional property files to read in. This list must be separated by spaces.
includeList	Returns the list of property files that were read in. This value is set by the PropertyReader class and cannot be overridden.
includeListCount	Number of property files read in. This value is set by the PropertyReaderclass and cannot be overridden.

Sample use of include is in MessageCode_en_US.properties file.

```
# List the other files to include as part of reading in
# this property file. Just the base name is needed.
# Must be space delimited only!
include = CoreResources
```

Advanced GUI Configuration

Laying Out Components

Layout values are based on Java's GridBagLayout component.

Modify Fill

The fill value is a string. When set to *BOTH*, the component will fill its entire x,y coordinates. When set to *NONE*, the component will fit only the area that it needs to. For example, if a button is set to *NONE*, then the button will only fill around the text. To be even more specific, if two text letters are on a button, then it will be smaller than if there are six text letters on the button.

Fill can also be specified to be *HORIZONTAL* or *VERTICAL* for specific fill in one direction. Note that for labels, fill should generally be set to *NONE*. If it is not *NONE*, then attempts to right-justify the label by setting the anchor to "*EAST*" will fail.

Modify Insets

Insets are given as four different values: top, bottom, left, and right. Each of these values will buffer a component from all other components. For example, if all of the values are 2, then the component will be two pixels on all four sides from the closest components.

Modify Weight

The weight is given as x and y values. The x stands for horizontal and y stands for vertical. The weight indicates how much to stretch the component relative to the other components on the frame.

Choosing the Font

Labels can have their font defined by the optional <Font> tag under the <LabelBehavior> tag.

```
<LabelBehavior>
  <Font name="Tahoma-BOLD-24"/>
</LabelBehavior>
```

Oracle Utilities Network Management System code uses the Java Font.decode() method (see the Java Font class documentation for further information).

To ensure that this method returns the desired Font, format the name parameter in one of these ways:

- fontname-style-pointsize
- fontname-pointsize
- fontname-style
- fontname
- fontname style pointsize
- fontname pointsize
- fontname style
- fontname

in which style is one of the four case-insensitive strings: "PLAIN", "BOLD", "BOLDITALIC", or "ITALIC", and pointsize is a positive decimal integer representation of the point size. For example, if you want a font that is Arial, bold, with a point size of 18, you would call this method with: "Arial-BOLD-18".

If a style name field is not one of the valid style strings, it is interpreted as part of the font name, and the default style is used.

Only one of ' ' or '-' may be used to separate fields in the input. The identified separator is the one closest to the end of the string that separates a valid pointsize or a valid style name from the rest of the string. Null (empty) pointsize and style fields are treated as valid fields with the default value for that field.

Some font names may include the separator characters ' ' or '-'. If str is not formed with three components (e.g., style or pointsize fields are not present in str) and fontname contains the separator character, then these characters may be interpreted as separators. In this case, the font name may not be properly recognised.

The default size is 12 and the default style is PLAIN. If the name does not specify a valid size, the returned Font has a size of 12. If the name does not specify a valid style, the returned Font has a style of PLAIN. If you do not specify a valid font name in the name argument, this method will return a font with the family name "Dialog".

Bold and Italic Labels - Labels can be defined as plain, bold, italic, or bold italic. This is done by the optional <Font> tag under the <LabelBehavior> tag.

This is an example of an italic label:

```
<Label name="LBL_ITALIC_TEXT">
  <LabelPlacement start="0,relative"/>
  <LabelBehavior>
    <Font style="ITALIC"/>
  </LabelBehavior>
</Label>
```

This is an example of a bold label:

```
<Label name="LBL_BOLD_TEXT">
  <LabelPlacement start="0,relative"/>
  <LabelBehavior>
    <Font style="BOLD"/>
  </LabelBehavior>
</Label>
```

This is an example of a label that is neither bold or italic:

```
<Label name="LBL_NORMAL_TEXT">
  <LabelPlacement start="0,relative"/>
</Label>
```

This is an example of a label that is both bold and italic:

```
<Label name="LBL_BOLD_ITALIC_TEXT">
  <LabelPlacement start="0,relative"/>
  <LabelBehavior>
    <Font style="BOLD ITALIC"/>
  </LabelBehavior>
</Label>
```

Advanced Configuration Options

This section describes components that provide more intricate configuration options.

JTable

1. **Column Editor** - A column in a table can be specified to have a different component for editing its cells. The valid components that can be specified are a ComboBox, a CheckBox, a TextField, or a TableCellTextArea. When a column has a different editor, such as a ComboBox, then all the rows in the table have a ComboBox for that column. A specific editor, rather than the default one, is generally specified when we want that column to be editable. When an editor is specified for a column, we should make sure that we provide all the necessary configuration options for that editor.

2. **Column and Row Popup Menus** - This option specifies the name of the right click pop up menus, which would show up when a user right clicks on a column header or one of the rows of the table. The name should be a valid name as per the name of the pop up menus that are already created while parsing the XML file.

3. **Status Keys** - The background and the foreground color of the rows in the table is configurable as per the contents of that row. The list of all the possible statuses for which we want the background and foreground colors to change are provided by status keys. The status keys are specific to the table and they should be valid values in a column of the data store from which the table obtains its data. This column is configured as the `status_column` in the `<TableBehavior>` element. Note that these row colors will be used to color the first visible column of the row if the user enables “Cell Coloring” instead of “Row Coloring” and no Cell Colors are configured for this table.

4. **Column Visibility** - the Column element allows a Visible sub-element with attributes for “initial” and “when,” which behave like the Visible elements available for other Components.

5. **Column Justification** - In tables, text is typically left justified and numbers are typically right justified. It is possible to override the justification on a per-column basis by using the justification attribute: `<Column key="EVENT_IDX" justification="`*`left`*`"/>`

   The options are:

   - `left`: the column is left justified.
   - `right`: the column is right justified.
   - `center`: the column is center justified.
   - `general`: numbers are right justified and other data is left justified. (Default)

6. **Text Wrapping** - To wrap text in a column, set the WrapText element to true:

```
<Column key="swmanStep.operationOutcome">
  <Editable initial="true"/>
  <WrapText initial="true"/>
  <Editor>
    <TableCellTextArea/>
  </Editor>
</Column>
```

   To make a wrapped column editable, use the TableCellTextArea editor:

```
<Column key="swmanStep.operationOutcome">
  <Editable initial="true"/>
  <WrapText initial="true"/>
  <Editor>
    <TableCellTextArea/>
  </Editor>
</Column>
```

7. **Preferred Column Widths** - To set columns within an auto-resized table to use a preferred column width, a minimum and max column width will need to be specified. Thus, the column can be resized within the limits of the minimum and maximum setting. When the table is initially displayed, it uses the preferred size, which is the existing "width" property setting.

   Example:

   XML - Table Behavior Definition

```
<TableBehavior auto_resize_columns="true"
data_source="DS_EXAMPLE">
  <Column key="Idx" />
  <Column key="Cls" />
  <Column key="Description" />
</TableBehavior>
```

   Property - Table Column Settings

```
TBL_EXAMPLE.Idx.text=Number
TBL_EXAMPLE.Idx.minimum_width=10
TBL_EXAMPLE.Idx.width=90
TBL_EXAMPLE.Idx.maximum_width=150

TBL_EXAMPLT.Cls.text=Type
TBL_EXAMPLT.Cls.minimum_width=10
TBL_EXAMPLT.Cls.width=90
TBL_EXAMPLT.Cls.maximum_width=150

TBL_EXAMPLT.Description.text=Description
```

   In this case, the table will be initially drawn with the first two columns having a width of "90" and the Description column spanning to utilize the rest of the space given to the JTable component. The first two columns can be resized, but only down to a width of 10 and up to 150. If the entire table is squished, the Description column will be cut down until all the columns have reached their preferred width. At which point all the columns will be squished at the same rate. Since the Description column does not have a preferred width, the width of the label ("Description") is used.

8. **Defining Column Headers** - A column header can be defined as either text or an icon. See the following example:

```
TBL_WA_ALARMS.STATUS.text = Status
TBL_WA_ALARMS.STATUS.icon = status.png
TBL_WA_ALARMS.STATUS.tooltip = Event Status
```

   The image file specified for an **`icon`** should exist in the tool's images configuration directory, along with all other image files.

   Define a **`text`** value for all column headers, including those defined as icons. The **`text`** value will be used in various dialogs where the column name is displayed.

   The **`tooltip`** is used to define a message that will pop up when the mouse is hovered over a column header. If an **`icon`** is defined, and a tooltip is not, the system will automatically use the **`text`** value as the tooltip.

9. **Defining Cell Colors** - If you do not wish to color your entire row, you can color individual cells using the `<CellColor>` element. The `check_column` is the column whose value will be compared, and the "key" will be the column whose color is changed. These cell colors are used both when Row Coloring is enabled and when Cell Coloring is enabled.

```
<CellColor check_column="FLISR_STATUS" key="EVENT_IDX">
  <Status key="-1" bg_color="yellow" fg_color="black"/>
  <Status key="1" bg_color="green" fg_color="black"/>
  <Status key="2" bg_color="pink" fg_color="black"/>
</CellColor>
```

Dialog Configuration Options

JBot dialogs are defined in xml files that start with DLG_. They have a similar configuration to JBot tools. One difference is that the dialogs use the datastores and jbot statuses of the tool they are attached to.

Here is an example of the start of a dialog configuration:

```
<JBotToolDialog width="280" height="120" modal="false"
  always_on_top="true">
```

- **`width`**: The width of the dialog.
- **`height`**: The height of the dialog
- **`modal`**: If the dialog requires the dialog be dismissed before using another part of the system, then this should be set to true.
- **`always_on_top`**: If a dialog should remain on top even if another window is selected, set this to true. Note that modal dialogs are implicitly always on top and do not need this attribute specified.

Performing Actions When Tools and Dialogs Open or Close

If a command or a list of commands needs to be run in response to a window action, such as a tool opening or closing, it can be defined using the `<ToolBehavior>` and `<DialogBehavior>` tags. These tags use a `<Perform>` subtag that takes a name and a category. The "name" attribute should be "Window" and the category name will be either `windowOpened` or `windowClosing`. `windowOpened` will allow the users to run code when the window opens for the first time. `windowClosing` will run when the users has requested that the tool close, but before the system actually closes the window (to allow the system to validate data, etc.). Other window events can also be caught. Please see the Java documentation for the WindowListener Interface for further information; the methods in that class can be used as the "category" attribute in this tag.

Here is an example on running a command when a tool opens:

```
<ToolBehavior>
   <Perform name="Window" category="windowOpened">
      <Command value="DoSomethingCommand"/>
   </Perform>
</ToolBehavior>
```

Here is an example of a command running when a tool closes:

```
<ToolBehavior>
   <Perform name="Window" category="windowClosing">
      <Command value="QuitCommand"/>
   </Perform>
</ToolBehavior>
```

Setting component height and widths normally, the size of the tool, along with the weight and fill attributes determine the size of the components. However, sometimes it is necessary to have a component be a certain size. To do this, specify a component_width and component_height attributes in the behavior tag. See the following example:

```
<Table name="TBL_WA_SUMMARY">
   <TablePlacement start="0,0" width="8" height="1" weight="1,0"
   fill="HORIZONTAL" insets="2,2,2,2" anchor="NORTHWEST"/>
   <TableBehavior data_source="DS_WA_SUMMARY" resize_columns="true"
   auto_resize_columns="false" component_height="59">
```

Calculated Fields

JBot has a rather complicated way of defining text substitution and formatting of fields. Normally, a component refers to a column as it exists in a datastore. (For example, DS_TABLE.code refers to the column code in the datastore DS_TABLE.) This section has examples of most of the different combinations that can be done with calculated fields. For each example, the field name and a sample output is given. The output uses the following datastore as its source:

Status	Priority	Code	Date
A	1	N	1/2/08 12:33
B	4	O	1/4/07 3:33
C	10		1/4/07 3:33
D	20	Q	1/4/07 3:33

Calculated fields are indicated by preceding the field with a #. The format is:

```
#field1, field2, …%[format definition]
```

The format definition is based on the java.text.MessageFormat class. (Please refer to the official Java documentation for more information on the MessageFormat class.)

Examples of Calculated Fields

Concatenating two fields together with a comma separating them:

```
#Status,Code%{0},{1}
A,N
B,O
C,
D,Q
```

Concatenating two or more fields together with a space separating them:

```
#Status,Code%{0} {1}
A N
B O
C
D Q
```

Replacing a field's value with another value:

```
#Status%{0}||A|Status 1|B|Status B
Status 1
Status B
C
D
```

Note: If a value isn't defined, then the original value is used unless the default value is provided. In the example above, if the value of the Status field is 'A' then it is replaced with 'Status 1' and if the value is 'B' then it is replaced with 'Status B'. Otherwise, it is unchanged. The default value is configured by adding a single value to the end of the list (`#Status%{0}||A|Status 1|B|Status B|Default`).

Replacing a field's value with a value from a property file:

Add the following lines to the JBotFormat_en_US.properties files:

```
STATUS.A = Status 1
STATUS.B = Status B
```

Then follow this example:

```
#Status%{0}|||STATUS
Status 1
Status B
C
D
```

Note: Default value can be configured by adding entry, which does not have the original value, to the property file. In the example above such an entry would be: `STATUS = Default Status`.

Replacing an integer code with a string :

```
#Priority%{0,choice,1#Priority A|5#Priority B|10# Priority C}
1, 4, 10, 20
Priority A, Priority B, Priority C, Priority C
```

Note that if the value is greater than the last choice, it will use the last choice. Likewise if a value is less than the first value, it will use the first value. Otherwise, it will use the largest lookup value that is not greater than the original value.

Performing a conditional:

```
#Status,Code,Priority %{0=B?1:2}
1
O
10
20
```

Performing a conditional if a value is null:

```
# Code,Status%{0=null?1:0}
N
O
C
Q
```

Displaying date and time fields:

Date, time, and date/time fields use the formatting defined in the Global_en_US.properties file. The following examples assume that the configured format is **MM/dd/yy HH:mm.**

Format	Value
Date%{0}	01/02/08 12:33
Date%(0,date}	01/02/08
Date%(0,date,long}	01/02/08 12:33
Date%(0,time}	12:33

How many % do I need?

A percent character (%) defines the start of a format.

A single % means that the original value should be used for both equality testing (such as cell filtering) and sorting. In other words if two source values are mapped to the same display value, then the underlying value will be used for things like cell filtering. If there is a unique mapping, however, performance is the best with this option.

A double (%%) means that the formatted value should be used for equality testing, but for sorting purposes the underlying value should be used. This would be appropriate for priority text strings. For example, if you had a priority field that got mapped to "Emergency", "Priority", "Routine", and "Planned," it would allow the sorting based underlying code instead of alphabetically.

A triple (%%%) means that the formatted value should be used both for equality testing and for sorting.

JBot DataStore Reference

To aid the implementer in determining the available columns for a **datastore**, it is possible to create a report that contains all the current values of a **datastore** for a running system. Because each system can have different configured columns, it is necessary to create this documentation from a running system.

Creating the JBot DataStore Report

1. Start the java application you wish to document. Ensure that the tools you are interested in are populated.
2. Bring up a shell window as the nms user on the nms server.
3. Type:

```
Action any.publisher* ejb jbot_report c:/OracleNMS/
datastore_report.txt
```

This will create the datastore report on all client machines that are logged in. If you wish to change the location that the report will be stored, change the above command.

Reading the Datastore Report

The report contains all the **datastore**s that are currently valid for the application, along with all of the valid columns.

See the following excerpt from the report, which describes the DS_WA_ALARMS **datastore** from Workagenda.

```
Datastore: :DS_WA_ALARMS
  CUSTOMER_NAME=
  COND_PHASES=B
  COND_STATUS_NAME=RDO
  EST_REST_TIME=07/27/09 14:02
  DEVICE_ALIAS=xfm_oh_JO-9976
  CTRL_ZONE_NAME_5=JO CO 9362
  CTRL_ZONE_NAME_6=
  CRIT_K=1
  .
  .
  .
```

Note that some of the columns listed are objects that would never be printed. For example, see the excerpt below:

```
  crew.zone_hdl=com.ces.corba.CES.Handle@1646de5
  crew.zone_hdl.class_number=4802
  crew.zone_hdl.app=0
  crew.zone_hdl.instance_number=1001094
```

The `crew.zone_hdl` is an object that would not be displayed. That object contains the class_number, instance_number, and app, which can be displayed.

JBot Tool Configuration

The `${CES_HOME}/dist/baseconfig/product/ops` folder contains sub-folders with each tool's configuration information. Tool folders typically contain the following sub-folders:

- **images**: contains images used by the tool
- **properties**: contains Java **`.properties`** files
- **xml**: contains **`.xml`** and **`.inc`** files

User Permissions

The USER_PERMISSIONS table is used to store the currently licensed products and is also used to grant Real Time and/or Study permissions to certain user types. Web Switching also utilizes this table to define access rights to certain elements within a switching sheet. This allows certain parts of a switching sheet to be edited by one user login type and not other login types.

The project licensed products will be listed in this table with an action value of 'LICENSED' and the user_name will have the licensed product name, like 'WebSwitching'.

For user type permissions within Web Switching, the user_name should indicate the user type and the action should be the actions that user type has permissions over within the sheet. For example:

```
INSERT INTO user_permissions  (seq_permission_id, user_name, action)
   VALUES( SEQ_USER_PERMISSION.NEXTVAL, 'Full Operations', 'ALL');
```

This would give the user of type "Full Operations" permissions over every aspect of a switching sheet. Use the "ALL" action to define this. Another example:

```
INSERT INTO user_permissions  (seq_permission_id, user_name, action)
 VALUES( SEQ_USER_PERMISSION.NEXTVAL, 'Switching Prep',
'Isolate_EXECUTE');
```

This gives the "Switching Prep" user type execute permissions for steps, but only when the steps are in an "Isolate" block. The format of the action string is "<Step Block Name>_<Step Action Name>".

The USER_PERMISSIONS table can also define what modes a user type has access to. In this case, any of the following actions can be defined for a user type:

- CREATE_STUDY_SESSIONS - Allow the user to create their own study sessions.
- USE_SHARED_STUDY_SESSION - Force the user to use a single shared study session called "SHARED_STUDY_SESSION". Normally study sessions are named based on the user's login ID. Thus, this ID cannot be given to a user for login purposes. Also, since this option forces this user or user type to share a single study session, then this user or user type should have limited control to the Control Tool or any actions that result in a change to the model.
- ACCESS_REAL_TIME - Allow the user to access the Real Time model.

For Product configured Web Workspace and Web Workspace with Web Switching:

- Start Real-time and Start Study mode buttons/menus are only visible when user has both CREATE_STUDY_SESSIONS and ACCESS_REAL_TIME permissions.
- Reset Study Session menu is only visible when user has CREATE_STUDY_SESSIONS permission.
- Close Study Session menu is only visible when user has both CREATE_STUDY_SESSIONS and ACCESS_REAL_TIME permissions.

Example:

```
INSERT INTO user_permissions  (seq_permission_id, user_name, action)

VALUES( SEQ_USER_PERMISSION.NEXTVAL, 'Switching Entry',
'ACCESS_REAL_TIME');
```

The "Switching Entry" users will not create study sessions and will be in a sense, always stuck in Real Time. If the user type is given access to a viewer and control tool, projects should consider using the USE_SHARED_STUDY_SESSION option instead.

These options are in place to minimize the demand on the services to keep track of individual study sessions used by users. The more study sessions that are active, the higher the burden on the services. Keeping the study sessions to a minimum is highly recommended and these rules should be used to do that.

Login Tool Configuration

JBot tools have a standard login tool that is used for all products. The login tool is responsible for determining which user type the user should log in as and verifying the password if LDAP integration is turned off.

To configure a tool to use the login tool, the **`product_name`** global property should be set in the tools configuration to the value as it exists in product column of the **`ENV_CODE`** table.

The current codes are:

- `CREW` (Web Workspace)
- `SWITCHING` (Web Request)
- `STORM` (Storm Management)
- `SERVICE_ALERT` (Service Alert)
- `OMS_CONFIG_TOOL` (Configuration Assistant)
- `WCB` (Web Callbacks)
- `WCE` (Web Call Entry)

The following example demonstrates configuring Web Call Entry (**`WCE`**) to use the login tool:

```
<JBotTool width="830" height="900">
   <GlobalProperties>
      <StringProperty name="product_name" value="WCE"/>
   </GlobalProperties>
...
```

User Session Configuration

The Login tool has a configuration option that handles user sessions when a user logs into the system from a different client.

Login Bean Properties

- **File**: ./jconfig/server/CentricityServer.properties

```
LoginEJB.force_relogin = <true | false>
```

When set to false, if a user is currently logged into the application (or has abnormally exited within two minutes), the user will receive an error message saying the user is already logged in. The user can be released using the Configuration Assistant to reset the login.

When set to true, if another login occurs for the same application and user, the original session will be automatically logged off. The system will not allow the user to save their work. The system will return a Dialog informing that the existing user was logged off, and that they should retry the login. Clicking the **Login** button again will then log the user into the system and the new user (session) will begin.

Master Window Configuration (Oracle Fusion Client Platform)

The main client application window for Web Workspace and Web Switching can be configured by using a separate XML file. Here is an example:

```
<dockingPositions xmlns="http://nms.oracle.com"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://nms.oracle.com http://
localhost/xml/docking.xsd"
             bounds="30,0,1220,942" maximized="false">
   <WEST floatSize="1003" dockSize="1210">
      <box>
         <leafBox height="300" width="400">
            <dockable dockWidth="400" floatHeight="300"
floatOrientation="-1" dockHeight="300"
ID="com.splwg.oms.client.crew.CrewIcons" floatWidth="400"/>
            <dockable dockWidth="400" floatHeight="300"
floatOrientation="-1" dockHeight="300"
ID="com.splwg.oms.client.authority.Authority" floatWidth="400"/>
         </leafBox>
      </box>
   </WEST>
   <EAST floatSize="180" dockSize="275">
      <box>
         <leafBox height="300" width="400">
            <dockable dockWidth="400" floatHeight="300"
floatOrientation="-1" dockHeight="300"
ID="com.splwg.oms.client.workspace.Workspace" floatWidth="400"/>
         </leafBox>
      </box>
   </EAST>
   <NORTH floatSize="180" dockSize="490">
      <box>
         <leafBox height="300" width="400">
            <dockable dockWidth="400" floatHeight="300"
floatOrientation="-1" dockHeight="300"
ID="com.splwg.oms.client.workagenda.WorkAgenda" floatWidth="400"/>
         </leafBox>
      </box>
   </NORTH>
   <SOUTH floatSize="180" dockSize="250"/>
</dockingPositions>
```

The file should be saved as the name of the application, with docking. For example, the name of the file for Web Workspace would be `Workspace_docking.xml`.

The easiest way to modify this file is to arrange the windows the way you wish, then exit the system. Bring up Windows Explorer, and locate `c:\Documents and Settings\[user]\.nms\system11.0.0.0.0\o.ide.11.1.1.1.33.53.67\windowinglayout.xml`

From that file, cut and paste the dockingPositions element to the configuration file. Note that the <dockingPositions> element should include the attributes listed above, although they are not in the `windowingLayout.xml`.

The “bounds” dockingPositions attribute specifies the position and size of the master frame. The numbers are the x, y, width, and height of the master frame.

The “maximized” attribute specifies if the frame should start maximized.

The default docking for applications that do not have a specific docking configuration is defined by jconfig/global/xml/JbotTool_docking.xml.

Work Agenda Configuration

Work Agenda configuration files are found in `${CES_HOME}/dist/baseconfig/product/ops/workagenda/`.

Example: Change the Heading of a Column

Scenario: You want to change a column heading from Feeder to Circuit.

Column headings are defined in `${CES_HOME}/dist/baseconfig/product/ops/workagenda/properties/`**`WorkAgenda_en_US.properties`** in the alarms panel section. You find the line:

```
################
# alarms panel #
################
...

TBL_WA_ALARMS.FEEDER_ALIAS.text = Feeder
```

1. Create a new blank **`WorkAgenda_en_US.properties`** file in the appropriate project or testing configuration directory (*e.g.*, `C:\OracleNMS\OPAL\jconfig\ops\workagenda\properties\WorkAgenda_en_US.properties`).
2. Add the line: `TBL_WA_ALARMS.FEEDER_ALIAS.text = Circuit`
3. In a terminal window, enter the following:

   ```
   cd c:\OracleNMS\OPAL\jconfig
   ```

4. Build the configuration using ANT:

   ```
   ant config
   ```

5. Log into Web Workspace to verify the column heading change. See **Testing the Java Client Configuration** on page 15-5 for details on configuring and starting Web Workspace in a testing environment.

Example: Adding a Column

Scenario: You want to add a column, MD, to the Work Agenda.

If the data is available to Work Agenda, but not visible, the column may be added to the configuration files. If it is a new column, it must be populated through a SQL command. This example assumes that the column exists.

1. Copy `${CES_HOME}/dist/baseconfig/product/ops/workagenda/xml/`**`WORKAGENDA_TBL_WA_ALARMS.inc`** to `C:\OracleNMS\OPAL\jconfig\ops\workagenda\xml\`.
2. Add a blank line following the line

   `<Column key="CTRL_ZONE_NAME_4"/>.`
3. On the blank line, add:

   `<Column key="mds_id"/>`
4. Open the project version of **`WorkAgenda_en_US.properties`**, created in the previous example.
5. Add the following lines:

   ```
   TBL_WA_ALARMS.mds_id.text = MDS Id
   TBL_WA_ALARMS.mds_id.width = 50
   ```
6. Build the configuration using ANT:

   `ant config`
7. Log into Web Workspace to verify the column has been added.

Crew Actions Configuration

The Crew Actions Tool configuration files are found in `${CES_HOME}/dist/baseconfig/product/ops/crew/`.

Crew Actions provides a mechanism for filtering crews based on control zone level; this filtering is capable of listening to the Work Agenda and filtering crews based on the event zone. The functionality is found in a Filter sub-menu description in **`CREW_ICONS_MENUBAR.inc`**:

```
<SubMenu name="MNU_FILTER_EVENT_ZONE">
    <RadioButtonMenuItem name="RBG_EVENT_ZONE_ALL" hide_icon="true"
     button_group="RBG_EVENT_ZONE"
     true_value="ALL" data_source="DS_EVENT_ZONE.LEVEL">
       <Enabled initial="true" when="ZONE_FILTER_ENABLED"/>
       <PressPerform>
           <Command value="FilterSelectedZoneCommand"/>
       </PressPerform>
    </RadioButtonMenuItem>
    <RadioButtonMenuItem name="RBG_EVENT_ZONE_DISTRICT" hide_icon="true"
       button_group="RBG_EVENT_ZONE" true_value="CTRL_ZONE_NAME_2"
       data_source="DS_EVENT_ZONE.LEVEL">
       <Enabled initial="true" when="ZONE_FILTER_ENABLED"/>
       <PressPerform>
           <Command value="FilterSelectedZoneCommand"/>
       </PressPerform>
    </RadioButtonMenuItem>
    <RadioButtonMenuItem name="RBG_EVENT_ZONE_OFFICE" hide_icon="true"
     button_group="RBG_EVENT_ZONE"
     true_value="CTRL_ZONE_NAME_3" data_source="DS_EVENT_ZONE.LEVEL">
       <Enabled initial="true" when="ZONE_FILTER_ENABLED"/>
       <PressPerform>
           <Command value="FilterSelectedZoneCommand"/>
       </PressPerform>
    </RadioButtonMenuItem>
    <Enabled initial="true" when="ZONE_FILTER_ENABLED"/>
</SubMenu>
```

Related files and settings:

- The Work Agenda WORKAGENDA_TBL_WA_ALARMS.inc includes the **FilterSelectedZoneCommand** that interacts with Crew Actions to apply the filter in Crew Actions when one (or more) Work Agenda row(s) is selected.
- The Crew Actions CREWICONS_TOOLBEHAVIOR.inc also includes the **FilterSelectedZoneCommand**.
- CREW_DATASTORES.inc contains the datastore class name for the event zone: <DataStoreClass name="DS_EVENT_ZONE"/>
- The CrewIcons.xml configuration file sets the tool behavior when filtered in the PNL_CrewFilters sub-panel.

Example: Remove the Event Filter Sub-menu from Crew Actions

Scenario: Your project does not require filtering by event zone.

1. Copy ${CES_HOME}/dist/baseconfig/product/ops/crew/xml/**CREW_ICONS_MENUBAR.inc** to C:\OracleNMS\OPAL\jconfig\ops\crew\xml\.
2. Remove the entire <SubMenu name="MNU_FILTER_EVENT_ZONE"> section.
3. Run `ant config`.
4. Log into Web Workspace. The sub-menu will no longer be included under the Filter menu.

Event Details Configuration

Event Details configuration files are found in ${CES_HOME}/dist/baseconfig/product/ops/eventdetails/.

Example: Add a New Drop-Down List

Scenario: You need to add a new drop-down list to the Event Details window. You will create a value called *animals* that will contain choices of *lions*, *tigers*, and *bears*.

Note that while the Configuration Assistant can add new values to an event details dropdown, it cannot create the new list. So for the first option, you need to add it directly to the database using SQL Developer or alternative SQL tool of your choice.

1. Add the following entry to the **PICKLIST_GUI** table.

```
Lion, animals_om, pushbutton, non_outage, 100
```

2. Add a column called **animals_om** (VARCHAR2(20)), to the **PICKLIST_INFO_UPD_TR** table.
3. Start Configuration Assistant. Select animals_om and add entries for Outage for lions, tigers, and bears. Select non-outage and add tigers and bears.
4. Copy the following files from: ${CES_HOME}/dist/baseconfig/product/ops/eventdetails/xml/

   to

   C:\OracleNMS\OPAL\jconfig\ops\eventdetails\xml\:

   - EVENTDETAILS_DATASTORES.inc
   - EVENTDETAILS_GLOBAL_PROPERTIES.inc
   - EVENTDETAILS_PNL_ACTIONS.inc

5. To the copied file, **EVENTDETAILS_DATASTORES.inc**, add the following datastore:

```
<DataStoreClass name="DS_ANIMALS_OM"
class="com.splwg.oms.client.eventdetails.PicklistGUIDataStore"
table="picklist_gui"/>
```

6. Next, in the **EVENTDETAILS_GLOBAL_PROPERTIES.inc** file, find the line that defines eventdetails.populate_datastores and add `DS_ANIMALS_OM` to the list.
7. Next, find the eventdetails.categories section and add `ANIMALS_OM` to the list.
8. In the **EVENTDETAILS_PNL_ACTIONS.inc** file, add the following section before the **LBL_WEATHER** definition:

```
<Label name="LBL_ANIMALS">
  <LabelPlacement start="2,0" height="1" width="1" weight="0,0"
   fill="NONE" insets="2,2,10,2" anchor="EAST"/>
</Label>
<ComboBox name="CMB_ANIMALS">
  <ComboBoxPlacement start="3,0" height="1" width="1"
   weight="1,0" fill="HORIZONTAL"/>
  <ComboBoxBehavior data_source="DS_EVENT_DETAILS.ANIMALS_OM"
   keys_data_source="DS_ANIMALS_OM.PANE_NAME"
   default_value="PROPERTY.UNSELECTED">
    <Editable initial="false"/>
    <Enabled initial="false" when="!USER_VIEW_ONLY"/>
    <SelectPerform>
      <Command value="SetStatusFlagCommand">
        <Config name="flag_names" value="EVENT_DETAILS_EDITED" />
        <Config name="flag_values" value="true" />
      </Command>
    </SelectPerform>
  </ComboBoxBehavior>
</ComboBox>
```

9. Create, or append to, the file **EventDetails_en_US.properties** with the following:

```
LBL_ANIMALS.text = Animals
```

10. Run `ant config`.
11. Log into Web Workspace. You should now see the new option when you bring up the Event Details window for an event.

Example: Add a Validation Rule

Scenario: You want to create a validation rule that requires the user to choose an animal in order to close Event Details.

1. Copy **EVENTDETAILS_VALIDATION.inc** from: `${CES_HOME}/dist/baseconfig/product/ops/eventdetails/xml/ to the project eventdetails/xml folder.`
2. Add the following line to the copied file:

```
<ValueCheck group_names="VERIFY_COMPLETE_FORM" check_type="value"
fail_type="fail" ignore_blank="false" match_on="false"
values="Unselected" prompt="Must fill in a value for Animals"
widget_name="CMB_ANIMALS"/>
```

Note that in the **EVENTDETAILS_MENUBAR.inc** file, there is the following section which runs the validation:

```
<Command value="RunValidationCommand" when="!SWITCHING_EVENT">
  <Config name="group" value="VERIFY_COMPLETE_FORM"/>
</Command>
```

Viewer Configuration

Viewer configuration consists of defining the various model layers and defining application properties that control the Viewer behavior. The Viewer configuration is read by the application server and, consequently, updates to Viewer configuration require restarting WebLogic to deploy.

Example: Add a Separate Layer for SCADA Fuses

Viewer configuration consists of defining the various device layers and defining various properties that control the Viewer behavior. The layer configuration is read by the application server.

The layer definitions are found in `${CES_HOME}/dist/baseconfig/product/ops/viewer/xml/`**`SPATIALLAYERS_LAYERS.inc`**.

Scenario: You want to remove all SCADA-controlled fuses from the Underground Fuses layer and add them to a new SCADA Fuses layer for display in the Viewer. (This scenario is based on the characteristics of the OPAL model.)

1. Copy **`SPATIALLAYERS_LAYERS.inc`** from: `${CES_HOME}/dist/baseconfig/product/ops/viewer/xml/ to the project viewer/xml folder.`
2. Search for the Underground Fuses layer definition:

```
<Layer name="Underground Fuses"
       active_on_start="true"
       screen_selectable="true"
       annotation_only="false"
       dxf_layer="true"
       condition_layer="false"
       electrical_layer="true"
       draw_order="6">
  <Class name="rack_fuse_ug_hd"/>
  <Class name="scada_fuse_ug_hd"/>
  <Class name="scada_rack_fuse_ug_hd"/>
  <Class name="fuse_ug_hd"/>
  <Class name="rack_fusr_ss_hd"/>
  <Class name="scada_fusr_ss_hd"/>
  <Class name="scada_rack_fusr_ss_hd"/>
  <Class name="fusr_ss_hd"/>
</Layer>
```

3. Copy the definition and paste the copy above the current definition. Edit the file to add a new SCADA Fuses layer definition and a modified Underground Fuses layer that does not include the SCADA fuse classes.

```
<Layer name="SCADA Fuses"
       active_on_start="true"
       screen_selectable="true"
       annotation_only="false"
       dxf_layer="true"
       condition_layer="false"
       electrical_layer="true"
       draw_order="6">
  <Class name="scada_fuse_ug_hd"/>
  <Class name="scada_rack_fuse_ug_hd"/>
  <Class name="scada_fusr_ss_hd"/>
  <Class name="scada_rack_fusr_ss_hd"/>
</Layer>
<Layer name="Underground Fuses"
       active_on_start="true"
       screen_selectable="true"
       annotation_only="false"
       dxf_layer="true"
       condition_layer="false"
       electrical_layer="true"
```

```
        draw_order="6">
    <Class name="rack_fuse_ug_hd"/>
    <Class name="fuse_ug_hd"/>
    <Class name="rack_fusr_ss_hd"/>
    <Class name="fusr_ss_hd"/>
</Layer>
```

4. Copy **`DLG_VIEWER_HIDE_DISPLAY_LAYERS.inc`** from: `${CES_HOME}/dist/baseconfig/product/ops/viewer/xml/ to the project viewer/xml` folder. Edit the file to add the new hide/display settings:
   - Add a check box to toggle this new layer.

```
<CheckBox name="CHK_SCADA_FUSES">
  <CheckBoxPlacement start="0,relative" weight="0,0"
   insets="0,0,0,0"/>
  <CheckBoxBehavior>
    <PressPerform>
      <Command value="ToggleLayersCommand">
        <Config name="layers" value="SCADA Fuses"/>
      </Command>
      <Command value="RedrawCommand"/>
    </PressPerform>
  </CheckBoxBehavior>
</CheckBox>
```

   - Label the button:

```
CHK_SCADA_FUSES.text =  SCADA Fuses
```

5. After changing the layers, you must delete the cached map files, which are located in $OPERATIONS_MODELS/ser. Delete all the files in the ser folder; they will be regenerated as the maps are reloaded.
6. Run **`nms-install-config --java`**
7. Restart WebLogic.
8. Start Web Workspace, open a Viewer, and start the Hide/Display tool to verify that SCADA fuses is listed as a layer.

Example: Change the Viewer Background Color

The Viewer's GUI configuration is defined in `${CES_HOME}/dist/baseconfig/product/ops/viewer/xml/`**`VIEWER_GLOBAL_PROPERTIES.inc`**.

Scenario: You want to change the background color of the Viewer drawing area.

1. Copy **`VIEWER_GLOBAL_PROPERTIES.inc`** from: `${CES_HOME}/dist/baseconfig/product/ops/viewer/xml/ to the project viewer/xml folder.`
2. Modify the viewer.background_color line from:

```
<StringProperty name="viewer.background_color"
 value="241,243,248"/>
```

to

```
<StringProperty name="viewer.background_color"
 value="black"/>
```

3. Run **`nms-install-config --java`**
4. Restart WebLogic.
5. Start Web Workspace, open a Viewer, and verify that the background color is now black.

User Type Configuration

There are two approaches to configuring user environments by user types. The first method creates project configuration directories for each specific user type/role/privilege; the second sets rules within configuration files to enable different tools or views based on the user type/role/privilege. The first method works best if there are big changes between the two versions; the second way makes more sense if there is just tweaks to the existing configuration.

In either method, user types must be defined in the **ENV_CODE** table.

User Role Configuration by Subdirectory

1. Create a subdirectory of the project configuration directory that would be used for a specific user type/role/privilege (for example, view-only).
2. Copy each .xml (or property) file, which needs to be different for that user type, into this directory. Everything would be at the same level in the subdirectory; in other words, the directory would contain all .xml, .inc, and .properties files that are specific for that user.
3. Edit the files to make the desired changes.
4. Have an entry in the **ENV_CODE** table for that user type and include the subdirectory name containing the configuration for the user type.

Example: Create a view-only subdirectory and change the viewer background color for them.

1. Create the view-only directory in the project configuration directory. For example: **$CES_HOME/*<project>*/jconfig/ops/view-only/**.
2. Copy **VIEWER_GLOBAL_PROPERTIES.inc** from: `${CES_HOME}/dist/baseconfig/product/ops/viewer/xml/` to the newly created view-only directory.
3. Modify the viewer.background_color line from:

```
<StringProperty name="viewer.background_color"
 value="241,243,248"/>
```

   to

```
<StringProperty name="viewer.background_color"
 value="0,0,0"/>
```

4. Run **nms-install-config --java**
5. Restart WebLogic.
6. Start Web Workspace, open a Viewer, and verify that the background color is now black.

User Role Constraints on Configuration

As in the other method, each application or tool that requires different access or a separate view for a user type will need to be configured for that user type.

1. If a project version of the tool configuration does not exist, copy it the appropriate project configuration directory.
2. You'll need to add restrictions, as appropriate, in the configuration files to turn on or off features for a user type.

Example: Restrict ability to create a switching sheet for view only user.

```
<Menu name="MNU_FILE">
 <SubMenu name="MNU_NEW">
   <MenuItem name="MNU_NEW_SHEET" icon="new.png"
    accelerator="control N" hide_icon="true">

     <Visible initial="false" when="!USER_VIEW_ONLY and
      DS_LOGIN_ENTRY.WEB_SWITCHING_ENABLED == 'true'"/>

     <Enabled initial="false" when="!USER_VIEW_ONLY"/>

     <PressPerform>
       <Command value="DisplayNewNMSDialogCommand"
        when="DS_LOGIN_ENTRY.ENV == 'WEB' and
        DS_LOGIN_ENTRY.TYPE == '&UserSwitchingEntry;'">
          <Config name="dialog" value="DLG_NEW_NMS_DIALOG"/>
          <Config name="check_authority" value="false"/>
       </Command>

       <Command value="DisplayNewNMSDialogCommand"
        when="DS_LOGIN_ENTRY.ENV == 'WEB' and
        DS_LOGIN_ENTRY.TYPE != '&UserSwitchingEntry;'">
          <Config name="dialog" value="DLG_NEW_NMS_DIALOG"/>
       </Command>
     </PressPerform>

</MenuItem>
...
</Menu>
```

Model Management Application Configuration

The Model Management application provides a user interface for viewing status of model build related events (full builds, patches, and pending maps) and initiating a model build scripts. The tool may be configured to run the model build scripts in the background or to run them synchronously and display the result in a dialog.

Scenario: You want to configure the Model Management application to rerun model build patches in the background.

- This scenario relies on the behavior of the BuildPatchCommand command. Whenever the command is called, you may pass the nohup command (to make the command run in the background) along with the script to run. Model Management has two files that utilize the BuildPatchCommand:
  - `DLG_BUILD_MAP.xml`: controls the actions when **Build Map...** is selected from the Model Management tool's **Action** menu.
  - `ModelManagement.xml`: the primary configuration file for the Model Management tool; includes the menu action for building a patch.

1. Copy ModelManagement.xml from: `${CES_HOME}/dist/baseconfig/product/ops/model_management/xml/` to the project `model_management/xml` folder.
2. Add nohup to the the MNU_RESUBMIT_PATCH popup menu command:

```
<PopupMenuItem name="MNU_RESUBMIT_PATCH"
 class="javax.swing.JMenuItem">
  <Enabled initial="false" when="TBL_PATCHES_SELECTED"/>
  <PressPerform>
    <Command value="BuildPatchCommand">
      <Config name="command" value="nohup ces_build_maps.ces"/>
      <Config name="arg_01" value="-noVerify"/>
      <Config name="datastore" value="DS_PATCHES"/>
    </Command>
  </PressPerform>
</PopupMenuItem>
```

Note: this example uses the standard ces_build_maps.ces script, but you may substitute a custom version by copying the script to the project scripts folder and modifying that file.

Feeder Load Management Configuration

Feeder Load Management Global Properties may be modified as follows:

- **`max_flm_tools:`** Set the maximum number of tools that may be started; default value is 2.
- **`refresh_minutes:`** Set the number of minutes before the calculations are refreshed; default value is 3 using the IntegerProperty. Fractional values are allowed using the DoubleProperty element; see example that follows.

Scenario: Configure Feeder Load Management to refresh every 90 seconds.

- The product configuration sets refresh_minutes using the IntegerProperty element. The refresh time for this scenario is equal to 1.5 minutes, a fractional time that is not valid for an IntegerProperty.

1. Copy **`FLMSummary.xml`** from: `${CES_HOME}/dist/baseconfig/product/ops/flm/xml/ to the project flm/xml folder.`
2. Open the new product version of FLMSummary.xml.
3. In the Global properties element:

```
<GlobalProperties>
    <StringProperty name="product_name" value="FLM"/>
    <IntegerProperty name="max_flm_tools" value="2"/>
    <!-- How often you allow the FLM Summary to automatically
         refresh, in minutes. -->
    <IntegerProperty name="refresh_minutes" value="3"/>
</GlobalProperties>
```

4. Change :

```
<IntegerProperty name="refresh_minutes" value="3"/>
```

to

```
<DoubleProperty name="refresh_minutes" value="1.5"/>
```

Right-To-Left Language Configuration

Application configuration for Right-To-Left languages, such as Arabic, is configured through the CentricityTool.properties and the ImageLocalize.properties files.

Note that symbology file tooltips may be localized by editing the **`.sym`** files. See **Chapter 8**, **Building the System Data Model**, for details on sym files.

CentricityTool.properties

Text direction is defined in `${CES_HOME}/dist/baseconfig/global/properties/`**`CentricityTool.properties`**.

1. Copy **`CentricityTool.properties`** from: `${CES_HOME}/dist/baseconfig/product/global/properties/` to the project `global/properties` folder.
2. Modify the text.direction line from:

   ```
   text.direction = LTR
   ```

   to

   ```
   text.direction = RTL
   ```

ImageLocalize.properties

The ImageLocalize.properties file is used to override the XML files that define images/icons. The file is located in `${CES_HOME}/dist/baseconfig/global/properties/`. The file has defined image files that will be flipped if RTL is set in CentricityTool.properties. You may also substitute other files by adding a substitution statement.

Scenario: You want to substitute an image for the info_ena.png () file.

1. Copy **`ImageLocalize.properties`** to the project `global/properties` folder.
2. Edit the file:

   ```
   # The following lists those file that should be flipped or changed
   # when displaying in another language.("flip" is used for "RTL"
   # with arrows)

   info_ena.png = your_file.png

   oracle/javatools/icons/navigateBack.png = flip
   oracle/javatools/icons/navigateForward.png = flip
   textBigger.gif = flip
   textSmaller.gif = flip
   ```

Customizing Applications

Applications may be extended by providing custom commands, which may then be configured as part of the application. Additionally, NMS commands may be called from an external systems.

This section contains the following topics:

- **Customization Examples**
- **Creating Custom Functions for Displaying Data**
- **Using Additional Libraries**
- **Invoking Commands from an External System**
- **Invoking Commands Using a Web Service**

Customization Examples

Prerequisites

This assumes the user is familiar with programming in Java and with Oracle Utilities Network Management System configuration.

The demo commands and tool are included as part of the OPAL configuration. Therefore, this documentation assumes that either the OPAL model is used, or all the demo tools and configuration are copied to the correct project directory.

Setup

To run these examples, the following should be added to WorkspaceMenuBarTool. This will add a button to Web Workspace to display the demo tool:

```
<MenuItem name="MNUITM_DEMO" hide_icon="true">
  <PressPerform>
    <Command value="DisplayToolCommand">
      <Config name="tool" value="DemoTool"/>
      <Config name="class" value="com.splwg.oms.jbot.JBotTool"/>
    </Command>
  </PressPerform>
</MenuItem>
```

Overview

The Demo Tool provides an example with various text fields, a table, and buttons that demonstrate how to integrate into an Oracle Utilities Network Management System application.

- The **Hello World** example displays a dialog.
- The **AddCommand** example adds two numbers and saves them in a third field.
- The **IncrementCommand** example shows how to access and change data.
- The **DemoFocusCommand** example shows how to call existing JBot commands.

The example code is in `$NMS_CONFIG/jconfig/java/src`. This is where any custom commands should be saved.

Access to data in Oracle Utilities Network Management System is saved in datastores, which are bound to the actual java swing components.

The demo tool is saved to `$NMS_CONFIG/jconfig/ops/test/xml/DemoTool.xml`.

Hello World

The Hello World example code provides a simple command to display a dialog box displaying text:

See $NMS_CONFIG/jconfig/java/src/demo/HelloWorldCommand.java:

```
package demo;
import com.splwg.oms.jbot.JBotCommand;
import javax.swing.JOptionPane;
public class HelloWorldCommand extends JBotCommand {
    public void execute() {
        JOptionPane.showMessageDialog(null, "Hello World!");
    }
}
```

AddCommand

The AddCommand example provides a command that reads two values from the system and saves the sum to a third value.

```
package demo;

import com.splwg.oms.jbot.JBotCommand;

import java.awt.AWTEvent;
import java.awt.Component;

import javax.swing.JOptionPane;

/**
 * This command adds two numbers
 */
public class AddCommand extends JBotCommand {
    public void execute() {
        // This parameter must exist or else an error will occur
        String var1 = getRequiredParameter("var1");

        // If this parameter does not exist, the value will be null
        String var2 = getParameter("var2");

        String result = getRequiredParameter("result");

        double retVal;
        try {
            String number1 = (String)getDataSourceValue(var1);
            retVal = Double.parseDouble(number1);
            if (var2 != null) {
                String number2 = (String)getDataSourceValue(var2);
                retVal += Double.parseDouble(number2);
            }
            setDataSourceValue(result, retVal);
        } catch (Exception e) {
            AWTEvent awtEvent = (AWTEvent)getJBotEvent().getEvent();
            Component component = (Component)awtEvent.getSource();
            JOptionPane.showMessageDialog(component,
            "Could not add the numbers", "Error",
            JOptionPane.ERROR_MESSAGE);
            setAbort(true);
        }
    }
}
```

IncrementCommand

The IncrementCommand example shows how to read and write to multiple rows in a datastore:

package demo;

```
import com.splwg.oms.jbot.IDataRow;
import com.splwg.oms.jbot.IDataStore;
import com.splwg.oms.jbot.JBotCommand;

import java.awt.AWTEvent;
import java.awt.Component;

import javax.swing.JOptionPane;

/**
 * This example show how to access and update a datastore that
 * has multiple rows.
 */
public class IncrementCommand extends JBotCommand {
    public void execute() {
        IDataStore ds = getDataStore("DS_DEMO_TABLE");
        synchronized(ds.getLockObject()) {
            for (IDataRow row : ds) {
                Integer count = (Integer) row.getValue("count");
            row.setValue("count", Integer.valueOf(count + 1));
}
        }
        ds.notifyObservers();
    }
}
```

DemoFocusCommand

The DemoFocusCommand is an example on how to call existing JBot commands from within a custom JBot command. The example can be used to focus on a device in the viewer:

```
package demo;

import com.splwg.oms.client.viewer.FocusOnHandleCommand;
import com.splwg.oms.jbot.JBotCommand;
import com.splwg.oms.jbot.JBotException;

import java.util.HashMap;

/**
 * This is an example of calling an existing JBot command. It will
 * focus on a device with a given handle, given a datastore
 * values of the class and index of the device handle.
 */
public class DemoFocusCommand extends JBotCommand {
    public void execute() {
      String dataSource = getRequiredParameter("handle");
      String handleStr = getDataSourceValue(dataSource).toString();
      int pos = handleStr.indexOf(".");
      if (pos == 0) {
          throw new JBotException("Invalid handle");
      }
      String handleCls = handleStr.substring(0, pos);
      String handleIdx = handleStr.substring(pos+1);

      HashMap map = new HashMap();

      map.put("handle_cls", handleCls);
```

```
        map.put("handle_idx", handleIdx);

    // the options to this are the command name, a map of parameters
    // (or null if it doesn't take parameters, and the source for this
    // command (which can normally be left as null)
        getEnv().getTool().getAdapter()
         .runCommand(FocusOnHandleCommand.class.getName(), map, null);
      }
}
```

Creating Custom Functions for Displaying Data

There are times when you want to display data that integrates information from another system, or format data in a way that calculated fields do not have the flexibility to display. In this case, calculated functions can be used.

The following is an example of calling a custom function:

```
<Column name="#ReverseFeeder"
definition="#demo.ReverseFormat(FEEDER_ALIAS)%{0}"/>
```

Normally, items to the left of the % are the list of columns. Custom functions take those columns and perform an operation on them. For example, the above call reverses the characters in the first column listed.

Function ReverseFormat Source

```
package demo;
import com.splwg.oms.jbot.CustomFormat;

public class ReverseFormat extends CustomFormat {

    public Object[] format(Object[] source) {
        String input = (String)source[0];
        char[] chars = new char[input.length()];
        for (int i=0; i < chars.length; i++) {
            chars[chars.length - 1 - i] = input.charAt(i);
        }
        source[0] = new String(chars);
        return source;
    }
}
```

See the javadocs for CustomFormat for more information.

Using Additional Libraries

If additional client libraries are needed, they should be saved to $NMS_CONFIG/java/lib. Any jar files in this directory will be unjarred, and included as part of nms_config.jar.

Invoking Commands from an External System

Commands can be invoked by sending high level messages, either by using the “Action” command or by using a web service. (It is recommended that the web service be used for production use).

A listener for a high level message is defined as follows:

```
<Perform name="HLM" category=”onMessage” type="DISPLAY_MESSAGE">
  <Command value="demo.DisplayMessageCommand"/>
</Perform>
```

This should be defined in the ToolBehavior portion of the tool you wish to integrate with.

The **type** is an arbitrary identifier of the action.

This can be invoked by running the following from the Oracle Utilities Network Management System server:

```
Action -add_soap USER.* DISPLAY_MESSAGE "Hello world"
```

USER should be replaced with the username of the nms user.

This configuration calls the following command:

```
package demo;

import com.splwg.oms.jbot.HLMEvent;
import com.splwg.oms.jbot.JBotCommand;

import java.util.List;

import javax.swing.JOptionPane;

/**
 * This displays a message to the user from an external system
 */
public class DisplayMessageCommand extends JBotCommand {

    public void execute() {
        HLMEvent hlmEvent = (HLMEvent) getEvent();
        List<String> args = hlmEvent.getMessage().getArgs();
        String message = args.get(0);
        JOptionPane.showMessageDialog(null, message);
    }
}
```

Invoking Commands Using a Web Service

This should be invoked by using the sendHLM webservice message.

The wsdl for the web service is located as follows:

- For Weblogic:

  `http://nms-server:7001/MessageBean/MessageBeanService?wsdl`

(Replace nms-server with the dns name or IP address of the Oracle Utilities Network Management System system to connect to.)

JBotCommand Methods Reference

These are commands that can be called from a JBot command:

getParameter

```
protected java.lang.String getParameter(java.lang.String key)
```

This returns the value of a configuration option for this command.

getDefaultmeter

```
protected java.lang.String getDefaultParameter(java.lang.String key,
                                               java.lang.String
defaultValue)
```

getBooleanParameter

```
protected boolean getBooleanParameter(java.lang.String key,
                                      boolean defaultValue)
```

getRequiredBooleanParameter

```
protected boolean getRequiredBooleanParameter(java.lang.String key)
```

getRequiredParameter

```
protected java.lang.String getRequiredParameter(java.lang.String key)
```

This returns the value of a configuration option for this command. If it does not exist, it throws a JBotException.

getParameterSubset

```
protected java.util.SortedMap<java.lang.String,java.lang.String>
getParameterSubset(java.lang.String prefix)
```

This will return the parameters in alphabetical order that start with the given prefix.

Parameters:

`prefix` - Prefix of the parameter to match.

Returns:

A sorted map of parameters

execute

```
public abstract void execute()
                      throws java.lang.Exception
```

This is the method invoked by the CommandProcessor when the Command is executed.

Throws:

`java.lang.Exception`

getName

```
public java.lang.String getName()
```

Returns command String key.

Returns:

```
java.lang.String
```

supressBusyCursor

```
public boolean supressBusyCursor()
```

Return True if this command should not display the hourglass. This should only be set to true if the command is very fast.

getEvent

```
public java.lang.Object getEvent()
```

Returns original event object. It could be any swing events for example.

Returns:

```
java.lang.Object
```

setStatusFlag

```
public void setStatusFlag(java.lang.String flag,
                          boolean status)
```

Set the specified status flag in the DataManager. These statuses determine validation, JButtons' enabled status, etc.

Parameters:

`flag` - the status value

`status` - the boolean status

getStatusFlag

```
public boolean getStatusFlag(java.lang.String flag)
```

Get the value of the specified status flag in the DataManager. These statuses determine validation, JButtons' status, etc.

Parameters:

`flag` - the status value

Returns:

True if flag is true, False if flag not found or flag is false.

fireStatusChanges

```
public void fireStatusChanges()
```

Notifies all interested Components that the Tool's statuses have changed.

getDataStore

```
public IDataStore getDataStore(java.lang.String dataStoreKey)
```

Returns the DataStore with the specified key.

Parameters:

`dataStoreKey` - the String key that describes the DataStore

Returns:

the DataStore

getCurrentDataRow

```
public final IDataRow getCurrentDataRow(java.lang.String dataStore)
```

A convenience method that will get the current datarow of a datastore.

Parameters:

`dataStore` - the name of the data store.

getJBotEvent

```
public JBotEvent getJBotEvent()
```

Returns JBotEvent object.

Returns:

com.ces.jbot.JBotEvent

isAbort

```
public boolean isAbort()
```

Indicates whether processing of additional commands in this package should be aborted.

setAbort

```
protected void setAbort(boolean b)
```

If true, instructs the command processor to not process any additional commands for this event.

getDataSourceValue

```
protected java.lang.Object getDataSourceValue(java.lang.String
dataSource)
```

Returns the value of a datasource in the form of [datastore].[column name].

Parameters:

`dataSource` - the datasource

Returns:

the value

setDataSourceValue

```
protected void setDataSourceValue(java.lang.String dataSource,
                                  java.lang.Object value)
```

Chapter 16

Control Tool Configuration

The intended audience for this chapter are project engineers or software engineers responsible for configuring the Oracle Network Management System (NMS) Control Tool. This chapter includes the following topics:

Overview

The Control Tool affects many different aspects of the NMS system including tools, such as Web Switching and Web Safety, as well as services, such as DDService, PFService, and SwService. Due to the interactions with the various components, the Control Tool configuration includes database table configuration as well as JBot XML configuration typical of the other Java-based tools.

Control Tool Configuration

Control Tool Database Table Configuration

Control Tool actions are defined in two NMS database tables. The **CONTROL_ACT** table defines the actions and the **CONTROL_AGGREGATES** defines the order of execution for multistep control sequences.

The Control Tool Workbook generates the `CONTROL_ACT` and `CONTROL_AGGREGATES` table insert statements; the SQL statements can then be added to your `<project>_control.sql` file. You can find the workbook at:

```
$CES_HOME/OPAL/workbooks/Oracle_Utilities_NMS_Control_Tool_1.11.xls
```

CONTROL_ACT Database Table Configuration

The CONTROL_ACT database table contains the definitions for each control action used in the Control Tool, as well as some actions used exclusively by Web Switching and Web Safety.

Definition

- act_key - a unique index for the record.
- act_cls - the action type for the action (see below)
- act_idx - the action identifier (see below)
- action_name - the name of the JBot Action to be executed for this record
- label - the label to display on the Control Tool
- instruct_label - the label for the instruct version of the action on the Control Tool, if desired
- switching_desc - the text displayed for the action in Web Switching steps
- switching_code - the short code used when entering manual steps in Web Switching
- description - the description displayed in Web Switching, User Log, and Event Log
- undo_act_key - the act_key of the undo action, used when creating go-back steps in Web Switching

Valid values for act_cls/act_idx pairs:

act_cls	act_idx	Description
CONDADD	tag, note, <condition name>	Add a condition of the passed type
CONDDLG	tag, note, <condition name>	Display the edit dialog of conditions of the passed type.
CONDREM	tag, note, <condition name>	Remove a condition of the passed type.
Commissioning	Action	A commissioning action (automatically added by the Commissioning Tool). Use the commissioning action in the action_name (WSW_STEP_COMMISSION, WSW_STEP_DECOMMISSION, WSW_STEP_UNDO_COMMISSION, WSW_STEP_UNDO_DECOMMISSION)
DDS	CLOSE, OPEN	Close or open the device.
DDS	EARTH, EARTH_DW	Place or remove an earth/ground on the device. If the device is a switch, the Control Tool will display a side selection dialog.
DDS	MOMENTARY	Create a momentary on the device.
FLISR	ISOLATE_RESTORE	A FLISR Isolate & Restore block (automatically added when creating a FLISR plan.
HLMsg	NOOP	Comment steps. Also used as the first step of an aggregate.

act_cls	act_idx	Description
JMS	<none>	An automatic JMService event step, used in the Event Log and User Log.
MTS	DISABLE_FLISR, ENABLE_FLISR	Disable or enable FLISR for the device.
Manual	NOOP	Manual steps.
NOOP	20, 30, <Number of seconds to wait>	Wait the specified number of steps. Used in FLISR to wait for SCADA responses.
SRS	PO_DOWN, PO_HERE, PO_UP	Move the outage downstream, to here, or upstream.
START	ControlEdit	Perform a model edit.
Safety	Action	Safety actions. Use the action_name column to specify the action (issue, unissue, release, complete, abort)
ScadaCtrl	<1 and the attribute number, 2 and the attribute number>	Send a SCADA control for the passed digital attribute. Use 1 + the attribute to clear, and 2+ the attribute to set. For example, for attribute 3 (AutoReclose), you would set the act_idx to 13 to clear AutoReclose, and 23 to set AutoReclose.
Switching	Block	A Switching block. Use the action_name to specify the type of block (WSW_BLOCK_CONSTRUCTION, WSW_BLOCK_CUSTOM, WSW_BLOCK_DEFAULT, WSW_BLOCK_FAULT_LOCATION, WSW_BLOCK_ISOLATE, WSW_BLOCK_MAINTENANCE, WSW_BLOCK_NOMINAL, WSW_BLOCK_RESTORE)

CONTROL_AGGREGATES Database Table Configuration

The CONTROL_AGGREGATES table lists the aggregate Control Tool actions in the order they are to be executed.

Definition

- parent_act_key – The first act_key, from the CONTROL_ACT table
- act_key – The nth act_key, also from the CONTROL_ACT table
- sequence_number – The order of the action

For example, if you had CONTROL_ACT records for:

- 100: HLMsg::NOOP, “Open & Tag”
- 200: DDS::OPEN, “Open”
- 300: CONDADD:tag, “Place Tag”

Then you would enter the following rows in your CONTROL_ACGGREGATES table:

- parent_act_key=100, act_key=200, sequence_number = 1
- parent_act_key=100, act_key=300, sequence_number = 2

The Control.xml File

Once you have defined the control actions, you need to specify which buttons to appear on the Control Tool for the device classes. You also need to map these buttons to the control actions that were defined in the CONTROL_ACT table, and you need to create JBot actions to match the `CONTROL_ACT.action_name` values.

Set up your `<Button>` or `<PopupMenuItem>` element like any other JBot button, but with a few important differences:

- Use the `data_source` attribute to list `"DS_LABELS.<the button name>"` or `"DS_LABELS.<the button name>:INSTRUCT"` to use the CONTROL_ACT.label or CONTROL_ACT.instruct_label.
- Set the <Visible> element based on the inheritance or the device class itself. It is recommended that you set up parent classes in your <project>_inheritance.dat for each logical grouping of device classes that will have different Control Tool options, then use those in these "`when`" clauses. For example:

  ```
  <Visible initial="false"
  when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS ==
  'control_tool_switch'}"/>
  ```

  ```
  <Visible initial="false"
  when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS ==
  'control_tool_breaker'}"/>
  ```

  ```
  <Visible initial="false" when="{DS_CONTROL_TOOL.DEVICE_CLASS in
  ('generator')}"/>
  ```

- Add `<ControlActions>` and `<ControlAction>` elements to list the CONTROL_ACT keys to use for each device class or group of device classes. List the actions in order and use the "`when`" clause so the Control Tool knows which CONTROL_ACT record you want to use for each device class. For example, you may configure different actions and button labels for an Open button (*Disconnect Generator*, *Disconnect Jumper*, *Open Switch*, etc.):

  ```
  <ControlActions>

      <ControlAction key="170" when="{DS_CONTROL_TOOL.DEVICE_CLASS in
      ('generator')}"/>

      <ControlAction key="210" when="{DS_CONTROL_TOOL.DEVICE_CLASS in
      ('inline_jumper','p_p_jumper','rack_sub_jumper','sub_jumper')}"/>

      <ControlAction key="580" when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS ==
      'switch'}"/>

  </ControlActions>
  ```

- List the JBot actions to perform in the `<PressPerform>` element. For operations and other actions you record in switching, you should always add an `ACT_BEGIN_ACTION` and an `ACT_END_ACTION` call to set flags, reset the control tool, and prepare it for the next user action.

  Note: buttons that only display other tools do not need the `ACT_BEGIN_ACTION` and `ACT_END_ACTION` actions.

  Pass the `$INSTRUCT_FLAG$` to the `ACT_BEGIN_ACTION` as true if this is an instruct button. Pass the `$SEND_TO_SWITCHING$` flag to the `ACT_END_ACTION` if this actions should be recorded in Switching or the Misc Log.

- List the JBot action you configured as the `CONTROL_ACT.action_name` between the `ACT_BEGIN_ACTION` and `ACT_END_ACTION` actions. For example:

```
<PressPerform>
    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_BEGIN_ACTION"/>
        <Config name="$INSTRUCT_FLAG$" value="true"/>
    </Command>

    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_OPEN"/>
    </Command>

    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_END_ACTION"/>
        <Config name="$SEND_TO_SWITCHING$" value="true"/>
    </Command>
</PressPerform>
```

Example

```
<PopupMenuItem name="BTN_INSTRUCT_OPEN_DEVICE"
class="javax.swing.JMenuItem"
data_source="DS_LABELS.BTN_INSTRUCT_OPEN_DEVICE:INSTRUCT">
  <Enabled initial="false" when="OPEN_DEVICE and
(DS_CONTROL_DEFAULT.CURRENT_MODE == 'RT')"/>
  <Visible initial="false" when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS
== 'switch'}"/>
  <ControlActions>
     <ControlAction key="170" when="{DS_CONTROL_TOOL.DEVICE_CLASS in
('generator')}"/>
     <ControlAction key="210" when="{DS_CONTROL_TOOL.DEVICE_CLASS in
('inline_jumper','p_p_jumper','rack_sub_jumper','sub_jumper')}"/>
     <ControlAction key="580"
when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS == 'switch'}"/>
  </ControlActions>
  <ValidValues>
    <RunGroup run_group="CHECK_OPERATION_TIME"/>
  </ValidValues>
  <PressPerform>
      <Command value="ExecuteActionCommand">
          <Config name="action" value="ACT_BEGIN_ACTION"/>
          <Config name="$INSTRUCT_FLAG$" value="true"/>
      </Command>
      <Command value="ExecuteActionCommand">
          <Config name="action" value="ACT_OPEN"/>
      </Command>
      <Command value="ExecuteActionCommand">
          <Config name="action" value="ACT_END_ACTION"/>
          <Config name="$SEND_TO_SWITCHING$" value="true"/>
      </Command>
  </PressPerform>
</PopupMenuItem>
```

Commonly Used Flags and Datastore Values

Commonly used flags and datastore values that can be used in when clauses:

- `OPEN_DEVICE/CLOSE_DEVICE` - open/close is a valid action based on the state of the device.
- `DS_CONTROL_DEFAULT.CURRENT_MODE` - "RT" (*i.e.*, real-time) or "STUDY"
- `DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS` - a set of all parent classes for the selected device.
- `DS_CONTROL_TOOL.DEVICE_CLASS` - the selected device class name
- `HAS_<condition class name>` - whether a condition of the class (capitalized) is active on the device. (Example: `HAS_INSTRUCT`, `HAS_TAG`, `HAS_NOTE`, etc.)
- `HAS_<condition>_<status 0-10>` - whether a condition with status 0-10 is active on the device. (Example: `HAS_INFO_0`, `HAS_HOLD_2`, etc.)
- `SCADA_OPERATED` - if there is SCADA telemetry on the device status point
- `HAS_<SCADA measurement>` - whether the device has a SCADA measurement (analog or digital) with the name (capitalized). (Example: `HAS_AUTORECLOSE`, `HAS_AMPS`, etc.)
- `<SCADA measurement>_ON/<SCADA measurement>_OFF` - whether the digital measurement is ON or OFF. (Example: `AUTORECLOSE_ON`, `AUTORECLOSE_OFF`, `FAULT_INDICATOR_ON`, etc.)
- `DS_LOGIN_ENTRY.WEB_SWITCHING_ENABLED` - "true" or "false"
- `FROM_SWITCHING` - whether the action is being instructed or completed from Web Switching
- `HIDE_CONTROL_TOOL` - option
- `INSTRUCT_ONLY` - whether the action being executed is an Instruct (as opposed to a Complete)

Optional Project Status Flags

- `HIDE_CONTROL_TOOL` - if set implemented in the project Control.xml and set to 'true,' this flag will close the Control Tool when another device is selected in the Viewer. If set to false, the Control Tool to remain open. This option is not included in the product version of Control.xml.

  To implement this option, add the following (with or without the comment at the beginning of the code snippet) to the project Control.xml file:

```
<!-- Control Tool window behavior. This is a configuration
option. Set this to 'true' to enable closing of Control Tool
when another device is selected in the Viewer. Set to false if
you want the Control Tool to remain open. -->

<Perform name="Window" category="windowOpened">
  <Command value="SetStatusFlagCommand">
    <Config name="flag_names" value="HIDE_CONTROL_TOOL"/>
    <Config name="flag_values" value="true"/>
  </Command>
</Perform>
```

Project_Control_Actions.inc Include File

The product **Control.xml** file includes the **CONTROL_ACTIONS.inc** file, which contains all of the product **`<Action>`** definitions. Project-specific actions should be defined in a **PROJECT_CONTROL_ACTIONS.inc** file.

> **Note:** You may find it useful to use the **CONTROL_ACTIONS.inc** as an example.

The following example illustrates how to define an action to add an Information tag. The condition class, **info**, is defined in the `CLASSES` table.

```
<Action name="ACT_ADD_INFO">
    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_ADD_CONDITION"/>
        <Config name="$CONDITION_CLASS$" value="info"/>
    </Command>
</Action>
```

Web Switching executes actions when you instruct or complete steps in Web Switching; therefore, if there is any validation needed to prevent execution or completion of steps based on device states, you should add it to the `<Action>` element, using *DisplayErrorCommand*. You may add any number of specific error messages.

If you wanted, for example, to enforce that the system can only place an informational tag on a device that has no active instructs, then you could add the following:

```
<Action name="ACT_ADD_INFO">
    <Command value="DisplayErrorCommand" when="HAS_INSTRUCT">
        <Config name="message_code" value="CANNOT_HAVE_INSTRUCT"/>
    </Command>
    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_ADD_CONDITION"/>
        <Config name="$CONDITION_CLASS$" value="info"/>
    </Command>
</Action>
```

And in MessageCode_en_US.properties, you would need the following:

```
CANNOT_HAVE_INSTRUCT=Cannot perform this action when an instruct
is present.

CANNOT_HAVE_INSTRUCT.title=Action Failed
```

Or you might only want to perform that check for instructs if the user is not instructing the current action:

```
<Action name="ACT_ADD_INFO">
    <Command value="DisplayErrorCommand" when="!INSTRUCT_ONLY
      and HAS_INSTRUCT">
        <Config name="message_code" value="CANNOT_HAVE_INSTRUCT"/>
     </Command>
    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_ADD_CONDITION"/>
        <Config name="$CONDITION_CLASS$" value="info"/>
    </Command>
</Action>
```

And if, for example, you also wanted to make sure the tool is in study mode, you could add another specialized message, either before or after the other message:

```
<Action name="ACT_ADD_INFO">
    <Command value="DisplayErrorCommand" when="!INSTRUCT_ONLY and
HAS_INSTRUCT">
        <Config name="message_code" value="CANNOT_HAVE_INSTRUCT"/>
     </Command>
    <Command value="DisplayErrorCommand"
      when="DS_CONTROL_DEFAULT.CURRENT_MODE == 'RT'">
        <Config name="message_code" value="CANNOT_USE_RT"/>
    </Command>
    <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_ADD_CONDITION"/>
        <Config name="$CONDITION_CLASS$" value="info"/>
    </Command>
</Action>
and:
CANNOT_USE_RT=Cannot perform this action in real-time mode.
CANNOT_USE_RT.title=Action Failed
```

Configuration Example: Adding an Undo Close Action

Adding the Undo Close function to the Control Tool requires modifications to the project `CONTROL_ACTIONS.inc`, `Control.xml`, and `Control_en_US.properties` files.

CONTROL_ACTIONS.inc

Add the ACT_UNDO_CLOSE action:

```
<Action name="ACT_UNDO_CLOSE">
   <Command value="OpenDeviceCommand">
      <Config name="work_deenergized" value="false"/>
      <Config name="trace_direction" value="2"/>
      <Config name="enable_dialog" value="true"/>
      <Config name="must_view_abnormals" value="false"/>
      <Config name="must_view_atos" value="true"/>
      <Config name="must_view_conditions" value="true"/>
      <Config name="show_breaker_info" value="true"/>
      <Config name="breaker_info_fields"
       value="FEEDER_ID_1,NOMINAL_VOLTAGE, INTERRUPTION_RATING"/>
      <Config name="no_phases" value="false"/>
      <Config name="undo_close" value="true"/>
   </Command>
   <Command value="ExecuteActionCommand">
      <Config name="action" value="OPERATE_DEVICE"/>
   </Command>
</Action>
```

Control.xml

Add the button/menu item to Control.xml.

> **Note:** the value of control action key will depend on the project's configuration.

```
<PopupMenuItem name="BTN_UNDO_CLOSE"
class="javax.swing.JMenuItem"data_source="DS_LABELS.BTN_UNDO_CLOSE">
   <Enabled initial="false" when="OPEN_DEVICE and !HAS_INSTRUCT"/>
   <Visible initial="false"
    when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS == 'switch'}"/>
   <ControlActions>
      <ControlAction key="6480"
       when="{DS_CONTROL_TOOL.DEVICE_CLASS_PARENTS == 'switch'}"/>
   </ControlActions>
   <ValidValues>
      <RunGroup run_group="CHECK_OPERATION_TIME"/>
   </ValidValues>
   <PressPerform>
      <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_BEGIN_ACTION"/>
        <Config name="$INSTRUCT_FLAG$" value="false"/>
      </Command>
      <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_UNDO_CLOSE"/>
      </Command>
      <Command value="ExecuteActionCommand">
        <Config name="action" value="ACT_END_ACTION"/>
        <Config name="$SEND_TO_SWITCHING$" value="false"/>
      </Command>
   </PressPerform>
</PopupMenuItem>
```

Control_en_US.properties

Add label and tooltip for the new button/menu item.

```
BTN_UNDO_CLOSE.text = Undo Close
BTN_UNDO_CLOSE.tooltip = Undo latest close operation for the selected
switch
```

Updating Control Tool Configuration in Production Systems

After a system is in production, Control Tool updates are typically applied for one or more of the following reasons:

- to provide control actions for new device classes that are being added to the network (see **Adding New Device Classes**).
- to map existing actions to existing device classes (see **Mapping Existing Actions to Existing Device Classes**).
- to add new actions (for existing device classes; see **Adding New Actions**).
- to change when an action is enabled (see **Changing When Actions are Enabled**).

Adding New Device Classes

Configure the new classes in your <project>_inheritance.dat to inherit from the correct superclass so that it automatically gets the desired Control Tool buttons.

Mapping Existing Actions to Existing Device Classes

Either change the inheritance of certain device classes to get the desired set of buttons, or change the <Visible> and the <ControlAction> elements in the buttons to include the added superclass.

Adding New Actions

Add a new CONTROL_ACT record and reference a new JBot Action name in it.Then use the existing PROJECTS_ACTIONS.inc examples as a guide and add the new JBot Action to it.

Also create the button in the Control.xml file, reference the CONTROL_ACT.act_key in it, and set up the <Visible> and <Enabled> tags.

Changing When Actions are Enabled

Modify the <Enabled> tag in the Control.xml file, as with the other JBot tools.

Aggregate, Secondary, or Associated Devices

If you have the AGGREGATE_DEVICES model table populated and you wish to operate devices from a different device's Control Tool, you will need to make use of the UseAggregateModelDeviceCommand and UsePrimaryDeviceCommand. Use the UseAggregateModelDeviceCommand and pass it the index of the aggregate to use and the subsequent operation Commands will operate the aggregate device. Use the UsePrimaryDeviceCommand to move control back to the selected device if you are performing aggregate actions.

Note that the Look Ahead does not take into account any previously instructed actions. So when you instruct an aggregate action on multiple devices, the Look Ahead results will not reflect the results of any previous actions.

Chapter 17

Web Switching Management Configuration

This chapter describes how to configure and administer Web Switching Management. It includes the following topics:

- **Configuring Classes and Inheritance**
- **Database Data Tables**
- **Database Data Tables**
- **Database Configuration Tables**
- **Global Web Switching Parameters**
- **GUI Configuration Overview**
- **Switching Sheets**
- **Switching Steps**
- **Web Safety**
- **High Level Messages**
- **Troubleshooting**
- **Installing the Web Switching BI Publisher Report Package**

Configuring Classes and Inheritance

Web Switching Management utilizes standard classes to define the switching sheet types. The following table lists the classes utilized by Web Switching Management:

Class Name	Purpose
switch_sheet_step	The class is used for switching step handles. This class is defined as part of the core classes and should not be changed.
switch_sheet_planned	The sheet class used for Planned switching sheet handles. This class is defined as part of the core classes and should not be changed.
switch_sheet_emergency	The sheet class used for Emergency switching sheet handles. This class is defined as part of the core classes and should not be changed.

Class Name	Purpose
switch_sheet_fault	This sheet class is not used by Product configuration, but it is defined as part of the core classes. This class can be redefined and given a new name if the project wants a new switching sheet type. The switching sheet class numbers are referenced in the SWMAN_SHEET_CLS database configuration table.
oc_switch_sheet	The sheet class used for Outage Correction switching sheet handles. This class is defined as part of the core classes and should not be changed.
flisr_switch_sheet	The sheet class used for FLISR switching sheet handles. This class is defined as part of the core classes and should not be changed.
switch_template	The sheet class used for Template switching sheet handles. This class is defined as part of the core classes and should not be changed.
switch_sheet	This class is used to give the Planned, Emergency, FLISR and Outage Correction sheet types their unique switching sheet numbers. The next_free_index value for this class in the CLASSES table defines the next available sheet number to use for these four sheet types. Since all four of these sheet types gather their switching sheet numbers from the same pool, none of them can have identical sheet numbers. For more information, see Sheet Types. For Product configuration, this class also is set up to inherit from classes switch_sheet_planned, switch_sheet_emergency and switch_sheet_fault. This inheritance defines whether events are associated to the steps recorded into the sheets. Events are associated to steps so that events follow the steps if they are moved from one sheet to another. If you define a new Planned or Emergency sheet type for your project, then you will need to add that new class to the list of classes that the switch_sheet class inherits from.
switch_misc	The sheet class used for the Miscellaneous Log handle. This class is defined as part of the core classes and should not be changed.
switch_sheet_safety	The sheet class used for stand alone safety documents. Stand alone safety documents are sheets behind the scenes with a web safety GUI front end. This class also triggers special processing throughout the system to process stand alone safety documents correctly. This class is defined as part of the core classes and should not be changed.
ss_isolate	This class inherits from the list of device class types that should be used to generate isolation steps for the Generate Isolate Steps button option found on the conductor based Control Tools. When looking for isolation points and a device of the inherited class type is found, then switching steps to open and tag the device will be generated. For more information, see Generate Isolation Steps.

Class Name	Purpose
ss_secure	This class inherits from the list of device class types that should be used to generate tagging switching steps for devices that are already open. When selecting the Generate Isolate Steps option on the conductor based Control Tool and a device of the inherited device class is traced to and found open, then it will be tagged. For more information, see Generate Isolation Steps.
safety_num_INFO	The safety document class used for INFO safety document handles. The safety document classes are referenced in the SWMAN_SAFETY_TYPES database configuration table.
safety_num_CLEAR	The safety document class used for CLEAR safety document handles.
safety_num_HOLD	The safety document class used for HOLD safety document handles.
safety_num_HOT	The safety document class used for HOT safety document handles.
safety_num_WARN	The safety document class used for WARN safety document handles.

Database Data Tables

The following database data tables are used by Web Switching Management to store data related to switching sheets and safety documents. Web Switching was designed in such a way that none of the data tables should need to be redefined by a project unless a field needs to be increased in size. The actual fields in each of the tables should not be altered by a project.

SWMAN_AUDIT_LOG. This table stores all the audit log entries for the switching sheets and safety documents. Safety documents also get their audit log entries from the SWMAN_STEP table. This would include when conditions are applied and removed for a document.

SWMAN_DELETED_CUSTOMER. This table stores a list of customers that were deleted from a sheet's impacted customer list. These customers are not actually deleted from the model. They are just marked as being removed from the impacted customer list.

SWMAN_IMPACTED_SUPPLY_NODES. This table stores the list of supply nodes impacted by a switching sheet's steps. In most cases, this list is generated manually by a user and is generated against the user's Study session.

SWMAN_PATCHES. This table will normally only ever have one record and that's the last model edit or build patch that was processed by the Web Switching service. The service determines the devices affected by the patch listed and flags any steps related to those devices. This table is used by internal processing and does not contain any data that may be displayed to a user.

SWMAN_SAFETY_DOC_EXTNS. This table stores the values of any entry fields configured on the safety document GUI that is not part of the base SWMAN_SAFETY_DOCS data table. This table is a key/value pair type of data table and may include values for comment fields, option menus and check boxes.

SWMAN_SAFETY_DOCS. This is the core data table for all the safety documents. This data table includes all the core information about the safety document like what state it is in, what sheet it is associated to, the crew is was issued to and whether it had been deleted or not.

SWMAN_SHEET. This is the core data table for all the switching sheets. This data table includes all the core information about the switching sheet like what state it is in, the sheet's version, the master device associated to the sheet, Start and Finish dates and other key elements pertaining to the sheet. The general rule is that if any value on the switching sheet has any code based processing, then it gets included in this table. Values that are just for display purposes would go into the SWMAN_SHEET_EXTN table.

SWMAN_SHEET_DOCUMENTS. This table stores all the external documents that have attached to the switching sheet. The documents are stored as BLOBs in this table. The table also includes a user description about the attachment, the file name and the size of the file.

SWMAN_SHEET_EXTN. This table stores the values of any entry fields configured on the switching sheet GUI that is not part of the base SWMAN_SHEET data table. This table is a key/value pair type of data table and may include values for comment fields, option menus and check boxes.

SWMAN_SHEET_EXTN_HIST. This table stores the current extension values for a sheet when the sheet is copied just before its version is incremented. This table is used to determine the differences between two switching sheet versions. Currently, there is no mechanism in place to display these differences to the user on the GUI. This table is being populated for reporting and diagnosis purposes only.

SWMAN_SHEET_HIST. This table stores a copy of the current sheet just before its version is incremented. This table is used to determine the differences between two switching sheet versions. Currently, there is no mechanism in place to display these differences to the user on the GUI. This table is being populated for reporting and diagnosis purposes only.

SWMAN_SHEET_VIEW_AREA. This table maintains the list of view areas that have been created and associated to each of the switching sheets.

SWMAN_STEP. This is the core data table for all the switching sheet steps. This data table includes all the core information about the switching sheet steps like what state the step is in, the sheet version the step was added under, the device associated to the step, and other key elements pertaining to the steps. The general rule is that if any value within the step has any code based processing, then it gets included in this table. Values that are just for display purposes would go into the SWMAN_STEP_EXTN table.

SWMAN_STEP_EXTN. This table stores the values of any entry fields configured within the switching sheet steps list that is not part of the base SWMAN_STEP data table. This table is a key/value pair type of data table and may include values for comment fields, option menus and check boxes.

Database Configuration Tables

The following database configuration tables are used by Web Switching Management to store configuration settings related to switching sheets and safety documents.

SWMAN_EVENT_ASSOC_TYPE. This table maps the event association types to names. Projects should only change these records if they wish to change the names of the associations.

SWMAN_SAFETY_TYPE_ACTIONS. This table maps the various types of step actions that can be associated to each safety document type. For instance CONDADD/hold actions can only be associated to HOLD documents, which DDS/OPEN operations can be associated to HOLD, CLEAR and HOT safety documents. This table controls these association rules.

SWMAN_SAFETY_TYPES. This table defines all the different types of safety documents configured for a project. Product configuration includes HOLD, Clearance, Informational, HOT and Warning safety document types. This table defines the following for each safety document type:

- The JBot tool panel and dialog that should be displayed when loading a safety document of this type.
- The numbering pool the safety document should use when generating unique document numbers. Product is configured to have each document type get their unique document numbers from separate numbering pools.
- The short description of each safety document type.

SWMAN_SHEET_CATEGORY. This table defines the list of sheet categories that every sheet type has to inherit from. Projects should not have any reason to alter the records in this table as they are pre-defined. The table should only be used to look up the description for each of the sheet categories. For instance, all sheets of category PLANNED will generate planned events when completing switching steps that impact customers. Each of the categories has pre-defined rules that define how the switching sheets should behave.

SWMAN_SHEET_CLS. This table defines the types of switching sheets configured for a project. This table defines the following for each switching sheet type:

- The sheet category the sheet type should inherit from.
- The JBot tool panel that should be displayed when loading a sheet of this type.
- The display order of the sheet types in the New Switching Sheet dialog.
- The numbering pool the sheet should use when generating unique sheet numbers. For instance Planned and Emergency sheets get their sheet numbers from the same numbering pool.
- The description of each sheet type. This description is displayed on the New Switching Sheet dialog.

SWMAN_STEP_STATE_MAPPING. This table is used by FLISR to map step state keys to a value of 0, 1, 2, or 16 where:

- **0** indicates that the step has no state
- **1** refers to any step in a completed state
- **2** is in reference to instructed states.
- **16** indicates that the step is deleted.

Global Web Switching Parameters

The following global Web Switching Management rules apply to all sheet types:

SwmanParameters.properties

Rule Name	Valid Values	Description
sheet.copy.num_types	Number	The number of sheet copy-clear field rules defined. The sheet.copy rules define the fields that should be cleared in a switching sheet when it is copied.
sheet.copy.type#.class	Number	The class of switching sheet that has sheet copy-clear field rules.
sheet.copy.type#.clear_fields	Comma Delimited List	A comma delimited list of core switching sheet field names that should be cleared when a switching sheet is copied. For example: completedDate.
sheet.copy.type#.clear_extns	Comma Delimited List	A comma delimited list of switching sheet extension field names that should be cleared when a switching sheet is copied.
sheet.copy.type#.clear_step_extns	Comma Delimited List	A comma delimited list of switching sheet step extension fields that should be cleared when a switching sheet is copied.
safety.copy.num_types	Number	The number of safety copy-clear field rules defined. The safety.copy rules define the fields that should be cleared in a safety document when it is copied.
safety.copy.type#.name	String	The safety document type that has safety copy-clear field rules.
safety.copy.type#.clear_extns	Comma Delimited List	A comma delimited list of safety document extension fields that should be cleared when a safety document is copied.
safety.copy.type#.stand_alone_keep_actions	Comma Delimited List	The list of control actions that should be kept when a safety document is copied as a stand alone safety document. For instance, safety documents have Place condition, Remove condition and safety state transition actions, but when copied only the Place condition actions are needed in the new stand alone safety document. Device operation (Open/Close) actions are prohibited. The format of the values are *<act_cls>:<act_idx>*. act_cls and act_idx pairs are a subset of the actions configured in the swman_safety_type_actions table. Example for a Hold Document: `CONDADD:hold, DDS:EARTH`

Rule Name	Valid Values	Description
step.openActKey	Control Tool Act Key	This act key is assigned to a step when an OPEN device operation message is processed by the Web Switching service. This can happen when a device is opened by a SCADA system.
step.closeActKey	Control Tool Act Key	This act key is assigned to a step when an CLOSE device operation message is processed by the Web Switching service. This can happen when a device is closed by a SCADA system.
step.crew.backToMasterDev	true/false	This option defines whether the crew should migrate back to the switching sheet's master device within the viewer when a step has been instructed and completed. If set to false, the crew will remain on the last instructed device after it has been completed.
STEP_STATE_SCADA_INSTRUCTED	Control Tool Act Key	This act key is used by internal processing to determine when a step has been SCADA Instructed.
STEP_STATE_INSTRUCTED	Control Tool Act Key	This act key is used by internal processing to determine if a step has been instructed.
STEP_STATE_COMPLETED	Control Tool Act Key	This act key is used by internal processing to determine if a step has been completed.
SWMANSHEET_TITLE_JBOT_CONFIG_VALUE	JBot Text Field Name	This field is used to populate the tab of each sheet. This field is normally hidden within the sheet Request and is a calculated field pulling values from multiple parts of the switching sheet.
SWMANSHEET_TITLE_JBOT_CONFIG_PARAM	JBot Flag Name	This is the flag to check to determine if the sheet has been edited or not. If it has been edited, then change the sheet tab to an italicized text.
sheet.requireFuzzyAuthority	true/false	If this parameter is set to true, then the user is required to take authority of the FUZZY zone to see switching sheets that are not associated to a modeled device. This parameter only comes into play when the environment is setup to filter switching sheets within the Open Switching Sheet list based on zones of authority.
STAND_ALONE_SAFETY_EDIT_FLAG	JBot Flag Name	This is the flag used to determine if the stand alone safety document has been edited or not. If it has been edited, then change the safety document tab label to an italicized text.

Rule Name	Valid Values	Description
SWMANSTEPS_JBOT_CONFIG_VALUE	JBot JTreeTable Name	This is the component name for the steps TreeTable.

sheetList.plannedAndExcludeDeletedAndOldSheets
 	
sheetList.excludeDeletedAndOldSheets
	String Value	These are **`where`** clauses that are added to the end of the query when gathering switching sheet data for the Open and New Sheet lists. The syntax is in JPQL. The query looks like this:
		
		`Select o from SwmanSheetView o`
		
		This is querying data from the swman_sheet table and swman_sheet_extn view.
		
		Example:
		`JOIN o.swmanSheetCls sheetCls JOIN sheetCls.sheetCategory sheetCategory WHERE sheetCategory.sheetCategoryName = com.splwg.oms.model.entity.swman.SwmanSheetCategoryName.PLANNED and (o.completedDate is null or o.completedDate + 60 >= CURRENT_DATE) and o.stateKey not in (255)`
		
		• Limit the Completed sheets to 60 days.
		• StateKey 255 = Deleted sheet state.
		• Limit the sheets list to only show PLANNED types.

PreLoadQueries = sheetList.excludeDeletedAndOldSheets	Comma Delimited List	PreLoadQueries is the comma-separated list of queries for sheets that the service will pre-load for performance reasons. These should be the **`where`** clauses listed above. For more information, see **Preload New and Open Switching Sheet Lists** on page 17-18.
AutoTransitionSCADASteps	true/false	This parameter defines whether instructed actions sent to SCADA should automatically complete the initiating switching step when the SCADA action is complete. Product has this value set to true.
disallowed_step_states	Comma Delimited List of Step State Keys	This is the list of states that will keep this rule from being applied. If any step is found with any of these states, then the sheet state action will not be applied.

Rule Name	Valid Values	Description
required_sheet_states	Comma Delimited List of Sheet State Keys	This is the list of sheet states that will allow this rule to be applied. This rule is normally used when there are no steps in the sheet and the state of the sheet has to be used instead. For instance, when removing steps from a sheet.
required_step_states	Comma Delimited List of Step State Keys	This is the list of states that will trigger this rule to be applied. If any step is found with any of these states, then the sheet state action indicated in this rule will be applied to the sheet.
sheet_state_action	Single Sheet State Action Name	This is the sheet state action name that should be applied to the sheet when the disallowed_step_states and required_step_states checks have passed. See database table TE_STATE_ACTIONS for a list of available actions configured where the APP field is equal to "WSW".
SHEET_STATE_TRANS.<Sheet Class #>.AuditLogType	String Value	This is the audit log type to be populated into the SWMAN_AUDIT_LOG database table. This value would be populated into the AUDIT_LOG_ENTRY_TYPE record field. For state transitions, this would normally be set to "STATE".
SHEET_STATE_TRANS.<Sheet Class #>.<Current Sheet State>.<New State Action Name>.AuditLogMsg = {SWMAN_AUDITLOG.STATE_CHANGE.text}	String Value	This is the audit log message to be populated into the SWMAN_AUDIT_LOG database table. This value will be populated into the LOG_COMMENT record field. This value can include a variable from the sheet tool's property file. In which case, the variable will be substituted for the value defined in the tool's property file. That same property value is normally used for state transition messages configured on the client side.

Understanding SwmanParameters.properties

For the sheet state transitions (**`SHEET_STATE_TRANS*`**), the **`#`** values can be a unique sequential number from 1 to N. The rules should be ordered in such a way that when the first rule is determined to be true, then the subsequent rules will be ignored. These rules are used to transition a switching sheet based on the step states. For instance, if a step transitions into the instructed state, then the state of the sheet is set appropriately as well.

Example: setting the sheet state to *In Progress* (Real-Time Instructed Steps) when an instructed step in status 310 is found.

```
SHEET_STATE_FROM_STEP.1.disallowed_step_states =
SHEET_STATE_FROM_STEP.1.required_step_states = 310
SHEET_STATE_FROM_STEP.1.sheet_state_action = instruct_step
```

If that rule does not pass, then the next rule is checked until one of the rules causes the sheet state to be updated. If no rule is applied, then no change will be made to the sheet's state.

Example: the sheet is transitioned to the *In Progress* (Real-Time Completed Steps) when there are no instructed steps and at least one step in the state indicated in the required_step_states parameter.

```
SHEET_STATE_FROM_STEP.2.disallowed_step_states = 310
SHEET_STATE_FROM_STEP.2.required_step_states = 265, 270, 280, 290, 320
SHEET_STATE_FROM_STEP.2.sheet_state_action = complete_step
```

Example: if the sheet is in the *In Progress* (Real-Time Instructed Steps) state and you don't have any instructed steps, then we transition to the *In Progress* (Real-Time Completed Steps) state.

```
SHEET_STATE_FROM_STEP.3.disallowed_step_states = 310
SHEET_STATE_FROM_STEP.3.required_sheet_states = 210
SHEET_STATE_FROM_STEP.3.required_step_states =
SHEET_STATE_FROM_STEP.3.sheet_state_action = complete_step
```

Example: in this scenario, we describe how to configure the audit log messages for the Emergency switching sheet when it goes into the *In Progress* state.

```
# Status 50 - New (Emergency)
# Status 220 - Hold/Pending
SHEET_STATE_TRANS.3109.AuditLogType = STATE
SHEET_STATE_TRANS.3109.50.instruct_step.AuditLogMsg =
{SWMAN_AUDITLOG.STATE_CHANGE.text}
SHEET_STATE_TRANS.3109.50.complete_step.AuditLogMsg =
{SWMAN_AUDITLOG.STATE_CHANGE.text}
# emerg_active is used when creating an Emergency sheet from an event.
SHEET_STATE_TRANS.3109.50.emerg_activate.AuditLogMsg =
{SWMAN_AUDITLOG.STATE_CHANGE.text}
SHEET_STATE_TRANS.3109.220.instruct_step.AuditLogMsg =
{SWMAN_AUDITLOG.STATE_CHANGE.text}
SHEET_STATE_TRANS.3109.220.complete_step.AuditLogMsg =
{SWMAN_AUDITLOG.STATE_CHANGE.text}
```

The Product Emergency switching sheet uses class number 3109. Specifying the current state of the sheet as part of the parameter name, is optional. If the current state is not given, then any time the sheet action instruct_step, emerg_activate, or complete_step are initiated, the audit log will be created. Since a sheet can jump between the states for the instruct_step and complete_step, we need to specify the from state value. Otherwise we would get a new audit log message each time a step in the sheet is instructed and completed.

Since we use variables for the AuditLogMsg's, the actual audit log message has to be looked up in the SwmanEmergencyTool_en_US.properties file. For Product this values is set to

```
SWMAN_AUDITLOG.STATE_CHANGE.text = Switching Sheet State Change
```

At this time, comments from other fields on the switching sheet cannot be incorporated into the server based audit log messages. That type of audit log message still has to be initiated from the client side through a JBot command called AddAuditLogDetailsCommand.

GUI Configuration Overview

The bulk of the Web Switching and Web Safety GUI configuration can be found in the jconfig/ops/webswitching directory. The configuration is spread across many files. This allows projects to customize bits and pieces of the configuration without having to define a custom version of the entire tool configuration. If at all possible, projects should inherit from Product as much as possible so that upgrades and patch installations are more easily applied to a project. The following tables describe the main modules used to configure the applications. Each of the configuration modules has an xml and properties file associated to it.

Web Switching

Name	Description
SwmanEntities.inc	This file includes a number of XML entities that are used throughout the Web Switching xml configuration files. The entities are used to give state key numbers readable names so that the configuration can be more easily followed. Instead of displaying a number in the configuration, a name is displayed. Changing the states and such to reference one entity also makes updating the configuration easier. If your project has defined a different switching sheet state for instance, then the single entity will only need to be altered instead of each instance where the entity is referenced.
SwmanBaseProperties	This module defines all the imports, datastores and dialogs used by each of the switching sheet types.
SwmanEmergencyTool, SwmanPlannedTool, SwmanMiscLogTool, SwmanTemplateTool, SwmanOutageCorrectionTool	Each of these modules defines the tool behavior for each of the switching sheet types. The modules also include all of the other modules used to build the GUI configuration for each of the switching sheet types.
SwmanToolBar	The switching sheet Menu/Toolbar configuration.
SwmanRequest	The switching sheet's Request tab configuration.
SwmanSteps	The switching steps Table configuration. Each of the step columns are defined in this module.
SwmanStepsPopupMenu	The right click switching Steps table context menu configuration.
SwmanStepsHeader	The switching steps toolbar button definitions.
SwmanHeader	The switching steps Event List and Crew List configuration.
SwmanEventsPopupMenu	The right click Events List table context menu configuration.
SwmanCrewsPopupMenu	The right click Crew List table context menu configuration.
SwmanImpactedCustomers	The switching sheet's Impacted Customers tab configuration.
SwmanImpactedCustomersPopupMenu	The right click Impacted Customers table context menu configuration.
SwmanSheetOverlaps	The switching sheet's Overlaps tab configuration.

Name	Description
SwmanExternalDocuments	The switching sheet's External Documents tab configuration.
SwmanExternalDocumentsPopupMenu	The right click External Documents table context menu configuration.
SwmanViewAreas	The switching sheet's View Areas tab configuration.
SwmanViewAreaPopupMenu	The right click View Areas table context menu configuration.
SwmanSafety	The switching sheet's Safety Documents tab configuration.
SwmanTracking	The switching sheet's tracking panel configuration on the Tracking/Audit Log tab.
SwmanAuditLog	The switching sheet's audit log panel configuration on the Tracking/Audit Log tab.
SwmanStatusBar	The switching sheet's status bar configuration.

Web Safety

Name	Description
SafetyBaseProperties	This module defines all the imports, datastores and dialogs used by each of the safety document types.
SafetyStateTransActions.inc	This file includes a set of actions pertaining to safety state transitions. The action names follow the format of *SAFETY_STATE_ACTION_<New State>.* These action names are looked up by the code, so the format of the action name has to be as specified. The actions each contain a list of commands that will be initiated when a safety document transitions to a new state. Each of the actions take one parameter called *$FROM_SWITCHING$*, which indicates whether the request is coming from Web Switching Management or not. See document for command `StateTransitionCommand` for more information. The file also includes an action defining the `actKey` to use when recording devices into the Tag Points device lists through view selections. This `actKey` pertains to the control action used to apply the appropriate tag to the device when the document is issued.
SafetyStateTransValidationRules.inc	This module defines the validation rules for the group name `CHECK_FOR_DEVICES_AND_CREWS`. This validation group is normally initiated when a safety document transitions to a new state. This can happen when executing steps from a sheet or from a safety document.
SafetyTool	This module defines the tool behavior for each of the safety document types. The modules also include all of the other modules used to build the GUI configuration for each of the safety document types.
SafetyTitle	The title configuration for all the safety document types.
SafetyToolbar	The safety document toolbar configuration.
SafetyBody	The document configuration for each of the safety document types. This module includes conditional checks for each of the safety document types to determine when to display components on the GUI as not all of the safety documents have the exact same GUI layout.

Name	Description
SafetyDeviceListPopupMenu	This is the popup menu in the Tag Points device list. It includes actions to remove, view and model verify the device.
SwmanStandAloneSafetyTool	Defines the tool behavior for the stand alone safety documents. This module also includes all of the other modules used to build the GUI configuration for this tool.
SwmanStandAloneSafetyMenuToolBar	Defines the MenuToolbar for the stand alone safety documents. This includes items like the Save and Exit options.
SwmanStandAloneSafetyPopupMenu	This is the popup menu in the Step Actions list. It currently only has one option and that's to verify individual steps when they have been impacted by a model edit.
SwmanStandAloneSafetyProperties	
SwmanStandAloneSafetyStatusBar	
SwmanStandAloneSafetyStepsToolbar	
SwmanStandAloneSafetyToolbar	

Switching Sheets

Sheet Types

Each of the switching sheet types are defined in the SWMAN_SHEET_CLS configuration table. Each switching sheet type has its own class. See Configuring Classes and Inheritance for further details on adding a class.

Within the SWMAN_SHEET_CLS configuration table, define which JBot tool configuration should be used when the sheet is loaded within the Web Workspace or Web Request environment. The switching sheet types can either share the same configuration or have their own. For instance, multiple Planned switching sheet types can all use the same SwmanPlannedTool definition and then within the tool configuration, define minor differences between the types based on the class of switching sheet being displayed.

State Transitions

State transitions for the switching sheets and their individual steps are all configured in the TE State Transition database tables where the *app* value to each of the tables is set to *WSW*.

> **Note:** See tables te_valid_states, te_status_groups, te_statuses, te_state_transitions, te_state_actions, te_expressions, te_init_state_rules, te_state_callbacks, and te_state_cb_args for more information.

Do not cross reference step and sheet states. Keep them completely separate. For example, create a state for the step Completed state and another state for the sheet Completed state. Do not try to use a single state for both the sheets and the steps.

Web Switching sheets support the following callbacks.

Callback Action Name	Description
safety_state_check	Determine if the sheet's associated safety documents are in the completed state. Switching sheets should not be completed when there are outstanding safety documents still issued to crews.
unrestored_pln_check	Determine if the switching sheet has any unrestored Planned events associated to it. In most cases, Planned switching sheets should not leave customers out of power.
create_switching_job	Create the Master switching sheet event that is normally used for Planned switching sheets.
complete_switching_job	Complete the Master and any Planned events associated to the switching sheet. This callback is normally used by Planned switching sheets.
validate_delegated_devices	Determine if the devices listed in the Delegated Zone (DCZ) document potentially isolate a section of the network.

Callback Action Name	Description
delegate	Delegate the DCZ document to a crew. Arguments: • CREW_NAME (required) - the field that maps to the crew data. Product uses "crew.crew_id", but other projects may map to an extn field, if this name is to be user-entered. • MUST_ISOLATE_FIRST (default: false) - whether the devices need to be open and isolated for the delegation to succeed.
undelegate	Release the delegated zone.

The following is an example for the Issue state:

```
INSERT INTO te_state_callbacks
  (app, cb_key, state_key, condition, action, abort_on_fail, can_undo,
   error_code)
VALUES
  ('WSW', 130, 232, 'PRE_ENTER', 'safety_state_check', 'Y', 'N', -
130);
INSERT INTO te_state_callbacks
  (app, cb_key, state_key, condition, action, abort_on_fail, can_undo,
   error_code)
VALUES
  ('WSW', 140, 232, 'PRE_ENTER', 'unrestored_pln_check', 'Y', 'N', -
140);
INSERT INTO te_state_callbacks
  (app, cb_key, state_key, condition, action, abort_on_fail, can_undo,
   error_code)
VALUES
  ('WSW', 160, 232, 'PRE_ENTER', 'complete_switching_job', 'Y', 'N', -
160);
```

The error_codes are used to display distinct dialog messages to the user when the action fails. The messages for these error codes are configured in the MessageCode_en_US.properties file.

The following is an example for error code "-130", which was referenced in the above te_state_callbacks example.

```
OmsClientException.WSW.STATE.CALLBACK.130 = Not all safety documents
are completed
OmsClientException.WSW.STATE.CALLBACK.130.title = State Transition
Failed
```

Sheet Data Fields

Data fields in this case are in reference to the fields found on the Request tab of the sheet. Data fields can be found anywhere on the sheet, but Product configuration has grouped the majority of them to one tab. The data fields are either stored in the SWMAN_SHEET or SWMAN_SHEET_EXTN tables.

The following are examples of how to reference a value from each of the tables.

SWMAN_SHEET

```
data_source="DS_SWITCHING_SHEET_LOCAL.deviceAlias"
```

For more information on the list of available data source values, refer to the DS_SWITCHING_SHEET datastore documentation.

SWMAN_SHEET_EXTN

```
data_source="DS_SWITCHING_SHEET_LOCAL.getExtnAttrByName.FEEDER_NAME.
stringValue"
```

For the switching sheet extension values, the attribute key name will be stored in the ATTRIBUTE_NAME field of the table record and from our previous example, the value will be stored in the STRING_VALUE field of that same record. The attribute name is case sensitive, so “feeder_name” does not map to the same value as “FEEDER_NAME.”

All fields not stored in the core SWMAN_SHEET table are stored as key/value pairs into this table. The application was designed this way so that fields can be added to the GUI without having to make database schema changes.

Open Switching Sheet List

The Open Switching Sheet list is populated through the DS_OPEN_SWITCHING_SHEET_TEMPLATE datastore, which is populated from the table SWMAN_SHEET and the database view SWMAN_SHEET_LIST_EXTN. The amount of data displayed in this list should be kept to a reasonable level. The more data that is displayed, the longer it will take the dialog to be displayed. For more information on limiting the amount of data queried from the database, see the Global Web Switching Parameters section.

Product has two clauses defined to limit the amount of data returned to the client to be displayed in the list. They are named sheetList.plannedAndExcludeDeletedAndOldSheets and sheetList.excludeDeletedAndOldSheets. Projects can define any number of these **`where`** clauses and have separately configured options on the Open Switching Sheet list to display different sets of switching sheets. The filters can also be used by different login environments to limit the switching sheet list based on type or even state.

The Open Switching Sheet list is initiated from the Web Workspace or Web Request Menu/Toolbar. The command that initiates that request is DisplayOpenNMSDialogCommand. This is the command that takes the name of the **`where`** clause you wish to use to gather the data from the database. Refer to the NMS Commands documentation for further details about this command.

Not only should the **`where`** clauses be used to limit the amount of data being passed to the client, but the database view SWMAN_SHEET_LIST_EXTN should also be defined with only the extension fields that are displayed on the Open Switching Sheet list. Query for data not displayed on the GUI is wasteful and should be avoided.

New Switching Sheet List

The New Switching Sheet type list is populated through the DS_SWITCHING_SHEET_CLS datastore, which is populated from the SWMAN_SHEET_CLS table. The pre-created sheet list displayed on this dialog is populated through the DS_OPEN_SWITCHING_SHEET_TEMPLATE datastore, which is populated from the table SWMAN_SHEET and the database view SWMAN_SHEET_LIST_EXTN. The amount of data displayed in this list should be kept to a reasonable level. The more data that is displayed, the longer it will take the dialog to be displayed. For more information on limiting the amount of data queried from the database, see the Global Web Switching Parameters and the Open Switching Sheet List sections.

The New Switching Sheet list is initiated from the Web Workspace or Web Request Menu/ Toolbar. The command that initiates that request is DisplayNewNMSDialogCommand. This command accepts a **`where`** clause name to use to gather the data from the database. The same **`where`** clauses used by the DisplayOpenNMSDialogCommand can be used with this command as well. Refer to the NMS Commands documentation for further details about this command.

Device to Sheet Operation List

The Device to Sheet Operation List is populated through the DS_DEVICE_SHEETS_LOCAL datastore, which is populated from the SWMAN_DEVICE_SHEET database view. The list displays the switching actions the device is associated to. The view is configured to gather all the steps associated to active switching sheets and to also pull in the last 30 days worth of completed actions as well. The filtering criteria is defined in the oracle view, which can be redefined by copying the view definition from the product_schema_web_swsheets.sql file and placing it into the project version of this file.

The Product dialog DLG_DEVICE_SHEETS has been configured to only pull in the sheet extension value called "Description." If projects require additional fields in this dialog, then they will have to create a custom project version of this dialog and add the necessary fields to the table defined in the dialog. The sheet extension fields have to also be included in the database view SWMAN_SHEET_LIST_EXTN. This view is also defined in the product_schema_web_swsheets.sql file. This is the same view that is used to populate the sheet extension values on the Open Sheet and New Sheet dialogs.

Preload New and Open Switching Sheet Lists

The PreLoadQueries SwmanParameters.properties parameter provides a comma-separated list of queries for sheets that the service will pre-load for performance reasons. The queries should be the **`where`** clauses used in the DisplayNewNMSDialogCommand and DisplayOpenNMSDialogCommand that take a long time to load and that display all types of sheets.

The caches will be pre-loaded and then augmented as sheets are created and edited. This should only be used for queries that include all sheet types and also include new sheets; **`where`** clauses not listed here will be queried from the database each time they are requested by the client.

If you have other **`where`** clauses or buttons on your dialogs that load only certain types of sheets, you should either:

a. Leave those queries off the PreLoadQueries list and continue to query the database each time you press those buttons.

   or

b. Reconfigure your dialogs and buttons so that you cache the large query above, retrieving the full set of data every time, but filter that data on your dialog.

To disable sheet caching, configure the parameter in your project property file with no argument. Like the following:

```
PreLoadQueries:
```

Model Verification

The Web Switching service initiates a query each time it receives a notification of a model build or edit. When this notification comes through, the following query is initiated:

```
SELECT sheet.switch_sheet_cls, sheet.switch_sheet_idx,
       step.step_cls, step.step_idx
FROM swman_sheet sheet, swman_step step,
     network_components nc, swman_patches sp
WHERE sheet.seq_sheet_id = step.seq_sheet_id AND
      // Exclude Block steps
      step.parent_step_id IS NOT NULL AND
      ( (step.dev_cls = nc.h_cls AND step.dev_idx = nc.h_idx) OR
        (step.gnd_node_cls = nc.port_a_cls AND
         step.gnd_node_idx = nc.port_a_idx) OR
        (step.gnd_node_cls = nc.port_b_cls AND
         step.gnd_node_idx = nc.port_b_idx) ) AND
      (nc.death > sp.patch_time OR  nc.birth > sp.patch_time) AND
      // Where the sheet and step are not in a termination state
      step.state_key NOT IN (<<List of terminal step states>>) AND
      sheet.state_key NOT IN (<<List of terminal sheet states>>) AND
     sheet.switch_sheet_cls not in (<<Outage Correction Sheet Types>>)
ORDER BY step.seq_sheet_id, step.step_idx
```

The MB_EDIT field in the SWMAN_STEP table is updated for each of the step records returned by this query. These steps will have to be validated by the user before switching sheet step executions can continue in the switching sheet.

Default Crews

To configure default crews for switching sheets, add the following sections to <project>_web_swsheets.sql:

```
DELETE FROM swman_crews WHERE seq_sheet_id = 0;
INSERT INTO swman_crews(seq_sheet_id, crew_key)
SELECT 0, crew_key FROM crews WHERE active='Y' AND
crew_name IN ('Default_Crew_1', 'Default_Crew_2');
```

Replacing '`Default_Crew_1`', '`Default_Crew_2`' with a comma-separated list of the names of the default crews as they exist in the crews table.

Versioning

Switching sheet versioning can occur manually or automatically. Product configuration is setup to automatically check in the switching sheet when it reaches the Issued state. This is done by initiating a call to the command CheckInSheetVersionCommand.

The version of a switching sheet will be automatically incremented when steps are manipulated (added, cut, pasted or deleted) within the sheet and when the switching sheet field CHECKED_IN has been set to *Y*. This field is stored in the SWMAN_SHEET table. The JBot flag VERSION_CHECKED_IN is set based on the value of the CHECKED_IN field. This flag is used by the JBot configuration to determine when to initiate commands based on version control.

Product configuration has been setup to increment the version automatically if any of the fields on the Request tab are altered. This is done by initiating the call to the command IncrementVersionCommand. This command will only execute if the switching sheet's CHECKED_IN database field is set to *Y*.

The current version of the switching sheet is stored in the REVISION field of the SWMAN_SHEET database table.

Overlaps

The switching sheet overlaps list uses the DS_OVERLAPS datastore. This datastore is populated from the SWMAN_OVERLAPS database view. The database view is defined in the product/sql/product_schema_web_swsheets.sql file. This same view is used by the Global Overlaps list, so any changes to this view will impact that list as well.

Product is configured to only include sheets classified under the category of PLANNED. The list is also filtered based on the state of the sheet. The list of state keys is included in the view definition. If any switching sheet states have been added to a projects configuration, this view may need to be redefined by the project.

External Documents

The switching sheet external documents list uses the DS_EXTERNAL_DOCUMENTS datastore. This datastore is populated from the SWMAN_SHEET_DOCUMENTS database table.

The External Documents functionality cannot be altered other than changing the column labels and sensitivity of the button options. The command DisplayFileChooserCommand, is used to gather files to be included in the list. Any changes to the file list are not saved to the database until the switching sheet is saved.

Generate Isolation Steps

The JBot command GenerateIsolateStepsCommand is used from the Control Tool to create a set of steps to isolate a piece of conductor within the model. The steps are generated based on the session the command was initiated from. If the Control Tool is in Real Time, then the Real Time model is used. If the Control Tool is in Study mode, then the user's study session is used. You also need to have a switching plan pre-created and in record mode in order to accept the generated steps. Both the session and the switching sheet requirements cannot be altered.

At this time, the command only supports isolating a conductor. The command uses the classes ss_isolate and ss_secure to determine what device types to create switching steps for. The command arguments determine the types of steps to generate for the isolate and secure device types. For more information, see the command documentation.

Switching Steps

State Transitions

State transitions for the switching sheet steps are all configured in the TE State Transition database tables where the *app* value to each of the tables is set to *WSW.*

> **Note:** See tables te_valid_states, te_status_groups, te_statuses, te_state_transitions, te_state_actions, te_expressions, te_init_state_rules, te_state_callbacks, and te_state_cb_args for more information.

Do not cross reference step and sheet states. Keep them completely separate. For example, create a state for the step Completed state and another state for the sheet Completed state. Do not try to use a single state for both the sheets and the steps.

Control Tool Actions

Web Switching Management uses the same rules that the Control Tool uses to determine if a control action is valid or not. Product configuration is configured to keep the two tools in synch. If the Control Tool does not allow an Open operation on a device, then a switching step with that same action will not allow the operation either. To get around this, the JBot flag FROM_SWITCHING can be used within the control/xml/Control.xml file and its include files to give actions alternate rules when the action originates from Web Switching Management. For more information, see the Control Tool Configuration chapter.

Step Columns

Switching step column data is either stored in the SWMAN_STEP or SWMAN_STEP_EXTN tables. Here are examples of how to reference a value from each of the tables.

SWMAN_STEP

```
key="swmanStep.comments"
```

For more information on the list of available data source values, refer to the DS_STEPS datastore documentation.

SWMAN_STEP_EXTN

```
key="swmanStep.getExtnAttrByName.details.stringValue"
```

For the switching step extension values, the attribute key name will be stored in the ATTRIBUTE_NAME field of the table record and from our previous example, the value will be stored in the STRING_VALUE field of that same record. The attribute name is case sensitive, so “Details” does not map to the same value as “details.”

All fields not stored in the core SWMAN_STEP table are stored as key/value pairs into this table. The application was designed this way so that fields can be added to the GUI without having to make database schema changes.

Device attributes (Addresses)

One specialized capability that the switching steps have not related to step execution is the ability to update device attribute information. When the command SaveAttributesCommand is called with a switching step extension field name, the value updated in the step for that device is propagated to the other steps in the steps list and is also passed to the device's associated attribute table. From this point on, when the device is used to record switching steps, the newly saved attribute information is displayed. In Product configuration, we utilize this feature for device address information, which is normally stored in the LOCATION field of the attribute tables. The location data is accessed through the database view ATT_ADDRESS. This view is model specific and has to be defined by each project. Here is an example of the view, which should be placed into the projects sql/<project>_schema_web_swsheets.sql file:

```
CREATE OR REPLACE VIEW att_address
    (h_cls, h_idx, att_name, att_value)
AS (
     SELECT h_cls, h_idx, 'location', to_char(location)
     FROM att_breaker where active = 'Y'
 UNION
     SELECT h_cls, h_idx, 'location', to_char(location)
     FROM att_bus_bar where active = 'Y'
 UNION
     SELECT h_cls, h_idx, 'location', to_char(location)
     FROM att_elbow where active = 'Y'
 UNION
     SELECT h_cls, h_idx, 'location', to_char(location)
     FROM att_fuse where active = 'Y'
 UNION
     SELECT h_cls, h_idx, 'location', to_char(location)
     FROM att_switch where active = 'Y'
);
```

Each project should add any additional device types that are configured to be included in recordable device operations.

The model attributes updated by Web Switching will be removed each time the attribute is updated from the GIS. This update can be setup to be ignored if a GIS update comes through with the old attribute value. In other words, we retain the attribute update from Web Switching as long as the GIS attribute value coming in is different. For more information, see chapter Building the System Data Model.

SCADA Auto-Transitioning

You can enable or disable auto-transitioning of SCADA switching steps using the *AutoTransitionSCADASteps* property, which is a property found in the `SwmanParameters.properties` file. If auto-transitioning is allowed, as it is in the product configuration, the property is set to *true*:

```
AutoTransitionSCADASteps = true
```

To disable auto-transitioning of SCADA switching steps, set the property to *false*:

```
AutoTransitionSCADASteps = false
```

Step Order Execution Rules

Each switching sheet type can be configured to force in-order step execution rules. When the rule is enforced, the Instruct, Complete, Abort and Fail options will only be enabled when all the steps prior to the selected step have been Instructed, Completed, Aborted or Failed. The same rule can also be applied for grouped steps. Normally, the sheets are configured so that grouped steps have the opposite rule applied to them so that users have the option of having some steps following the rule and others more relaxed.

The two rules are each defined in the Swman<Sheet Type>Tool.xml files. They are:

```
<BooleanProperty name="out_of_order_execution" value="false"/>
<BooleanProperty name="group_out_of_order_execution" value="true"/>
```

`out_of_order_execution:` When set to true, steps in a switching sheet can be Instructed, Completed, Aborted and Failed in any order. For Product configuration, this is set to True for Emergency switching sheets. Planned, Outage Correction and Template sheets have this option set to false.

`group_out_of_order_execution:` When set to true, steps in a step grouping can be executed out of order. If out_of_order_execution is set to true, then any steps listed prior to a grouping have to be completed before these steps can be completed out of order. For Product configuration, this option is set to true in Planned and Template sheets. It is set to false in Emergency and Outage Correction sheets.

To alter the step execution rules for a sheet type, simply alter these values in your project version of the `Swman<Sheet Type>Tool.xml` file.

Web Safety

State Transitions

State transitions for the safety documents are all configured in the TE State Transition database tables where the *app* value to each of the tables is set to *SF.*

> **Note:** See tables te_valid_states, te_status_groups, te_statuses, te_state_transitions, te_state_actions, te_expressions, te_init_state_rules, te_state_callbacks, and te_state_cb_args for more information.

Web Safety supports the following callbacks:

Callback Action Name	Description
check_safety_crew	Determine if a crew has been assigned to the document. This check is optional and can be configured to cause the transition to fail if a crew is not assigned. For Product configuration, we also have a validation rule setup to do the crew check on the client. This is done by calling the validation group `CHECK_FOR_DEVICES_AND_CREWS`.
update_safety_conditions	This action is used to update the status of the conditions associated to the safety document. This action requires an argument called `STATUS`. The status argument takes a number value. A status value of zero returns the condition back to normal so that it can be manipulated by the Control Tool. The condition cannot be removed when its status is in anything other than status zero. The status value of the condition can be used to change the symbol of the condition within the viewer.

The following is an example for the Issue state:

```
INSERT INTO te_state_callbacks
  (app, cb_key, state_key, condition, action, abort_on_fail,
error_code)
VALUES
  ('SF', 100, 110, 'PRE_ENTER', 'check_safety_crew', 'Y', -120);
INSERT INTO te_state_callbacks
  (app, cb_key, state_key, condition, action, abort_on_fail,
error_code)
VALUES
  ('SF', 110, 110, 'PRE_ENTER', 'update_safety_conditions', 'Y', -
100);
INSERT INTO te_state_cb_args
  (app, cb_key, arg_key, arg_name, arg_value)
VALUES
  ('SF', 110, 100, 'STATUS', '1');
```

The error_codes are used to display distinct dialog messages to the user when the action fails. The messages for these error codes are configured in the MessageCode_en_US.properties file. Here is an example for error code "-120", which was referenced in the above te_state_callbacks example.

```
OmsClientException.SF.STATE.CALLBACK.120 = Safety document has to be
assigned to a crew
OmsClientException.SF.STATE.CALLBACK.120.title = State Transition
Failed
```

Safety Document Data Fields

Data fields in this case are in reference to the fields found on the safety document that are not being pulled from the associated switching sheet. Data fields can be found anywhere on the safety document. The data fields are stored in the SWMAN_SAFETY_DOC_EXTNS table. Here is an example of how to reference a value from that table.

```
data_source="DS_SAFETY_DOCUMENT_LOCAL.doc.getExtnAttrByName.DESCRIPTIO
N.stringValue"
```

For more information on the list of available data source values, refer to the DS_SAFETY_DOCUMENT datastore documentation.

Configuring Stand Alone Safety Documents

For the most part, stand alone safety uses the existing configuration from the standard safety documents that are associated to switching sheets. There are a few additional configuration items that require additional attention when dealing with stand alone safety documents.

Safety Control Actions

The configuration table **SWMAN_SAFETY_TYPE_ACTIONS** plays a critical role in distinguishing key aspects that should only pertain to stand alone safety documents and not to regular safety documents associated to switching sheets. The new columns that were added to this configuration table include:

- **STAND_ALONE** - This indicates whether the action is allowed in a stand alone safety document. If a value other than 'Y' is specified for the action record, then the action will generate an error if the user attempts to paste this type of action into the stand alone safety document. During control tool recording, if the Real Time action being recorded is not found in this list, then no error is generated and the action is sent to the Miscellaneous log. Open and Close actions are prohibited from being associated to stand alone safety documents. Stand alone safety documents do not have event lists and cannot process event related processing.
- **ACTION_MAP_NUMBER** - This number is used to map key actions when safety documents are copied. For instance when copying a HOLD document as a Clearance document, the CONDADD:hold actions are all replaced with CONDADD:clear actions. This field defines that mapping in how the actions are replaced.

Flagging Safety Modifications

Stand alone safety uses a status flag called SAFETY_EDITED. This flag is defined in the SwmanParameters.properties file under the parameter **STAND_ALONE_SAFETY_EDIT_FLAG**. This flag determines when the safety documents tab label should be italicized. This flag should be set when a field on the safety document is edited by the user. This can be done like the following:

```
<TextAreaBehavior rows="4"
data_source="DS_SAFETY_DOCUMENT_LOCAL.doc.getExtnAttrByName.NON_MODELE
D_DEVICES.stringValue"  modify_flag="SAFETY_EDITED">
```

High Level Messages

The Switching Service (SwService) is used to process FLISR switching requests and also accepts the following High Level messages:

```
Action any.SwService <command> <arguments>
```

Where:

Command	Arguments	Description
debug	<N>	Sets the debug level: • 0 = Debug off • 1 = Debug on • 2 = Further details about database queries • 3 = Full debug
relock [Sheet Handle]		When no argument is given, then unlock all the switching sheets and send a request to each of the clients asking them to reestablish their single user switching sheet locks. This command can be used to clear up any orphaned locks that may still be active after an application lost network connectivity or crashed. When a switching sheet handle in the form of "<Sheet Cls>.<Sheet Idx>" is given, then only that one sheet is unlocked.

Troubleshooting

Through high-level Action messages debug can be turned on or off for parts of Web Switching Management. The debug categories can be used to debug configuration issues as well as runtime issues.

Web Switching Management debug category names:

Category Name	Debug Description
DELEGATE	Delegation actions, including validating isolation devices.
HLMESSAGE	Not just for Web Switching, but this debug category displays debug about High Level message processing.
IMPACTED_CUSTOMERS	Impacted Customers.
LOCK_OBJECT	Sheet Locking and Unlocking.
SAFETY	Safety documents.
SHEET	General debug category wrapped around most actions pertaining to a switching sheet.
SHEET.EVENT_ASSOC	Event associations.
SHEET.REVISION	Switching sheet revisions.
STEP	General debug category wrapped around most actions pertaining to a single switching step.
STEP.EXECUTE	Step Executions.
STEP.REVISION	Switching sheet step revisions.
STEPS	General debug category wrapped around most actions pertaining to the switching steps.
STEPS.EDIT	Step editing.
VALIDATION	Validation rules.

To turn on debug for a category:

```
Action any.publisher ejb debug <Category Name>=1
```

To turn off the messages:

```
Action any.publisher ejb debug <Category Name>=0
```

To turn on and off debug for all categories:

```
Action any.publisher ejb debug 1
Action any.publisher ejb debug 0
```

Installing the Web Switching BI Publisher Report Package

The Oracle Utilities Network Management System product configuration has a packaged version of Web Switching reports that may be used to preview and print switching sheets, safety documents, and the Miscellaneous Log. The package is delivered in the `nms_configuration.zip` file, which extracts to:

```
$NMS_CONFIG/jconfig/ops/bi_publisher/WebSwitching
```

This folder contains archive files that must be uploaded to BI Publisher Catalog. The next set of steps will guide you through this process:

Default Installation

Following are installation steps when parameter 'WEB_bipub.reportFolder' is not set.

1. Log into BI Publisher (`http://`*`<BIP server name>`*`:9704/xmlpserver/`) as the Administrator from a browser that has access to the WebSwitching folder. The files it contains can be copied from its default installation directory to a PC of your choice.
2. Set up a database connection by going to the Oracle BI Publisher Administration page, navigating to the **JDBC Connection** page under the Data Sources section, and then click **Add Data Source**.
3. In the **Name** field, enter `WebSwitching`.
4. Set the **Driver Type** to Oracle 9i/10g/11g.
5. Set the **Database Driver Class** to oracle.jdbc.OracleDriver.
6. Set the **Connection** string to: `jdbc:oracle:thin:@`*`<your machine>`*`:1521:`*`<the ORACLE_SID>`*
7. Set the **username** and **password** to match the your Oracle Utilities Network Management System database login values.
8. Click **Test Connection** and verify that it is properly configured.
9. Click **Apply.**
10. From the **BI Publisher Catalog** page, select **Shared Folders** from the folders tree.
11. On top of the folders section, click the **New** drop down and select **Folder** from the list.
12. Enter `WebSwitching` as the folder name, then click **Create**. The new folder is added.
13. Click to open the **WebSwitching** folder.
14. On top of the folders section, click the **New** drop down and select **Folder** from the list.
15. Enter `Resources` as the folder name, then click **Create**. The new folder is added within the WebSwitching folder.
16. Expand the **Resources** folder.
17. In the task section on the bottom left of the page, click **Upload**.
18. Browse to the `WebSwitching/Resources` report package folder and upload the archive WebSwitching.xsbz, which contains the sub-template file.

    Note: when uploading, select the “Overwrite existing report” option followed by the **Upload** button.

19. Navigate back to `Shared Folder/WebSwitching` in the catalog page. Upload the following files from the WebSwitching report package:
    - WebSwitching.xdmz
    - WebSwitchingMiscLog.xdmz
    - PlannedSheet.xdoz
    - OutageCorrectionSheet.xdoz
    - TemplateSheet.xdoz
    - StandaloneSafety.xdoz
    - MiscLog.xdoz

You should now be able to preview and print reports from Web Switching and the Miscellaneous Log.

Multiple Environment Installation

It is recommended that you first install Web Switching reports for single environment before beginning a multiple environment installation.

CES_PARAMETERS Configuration

Configure the following parameters in the CES_PARAMETERS table.

CES_PARAMETERS Attribute	Description
WEB_bipub.JDBCDriverClass	The JDBC driver class for the data source, for example, *`oracle.jdbc.OracleDriver`* (the default value).
WEB_bipub.JDBCURL	The connection string for the data source, jdbc:oracle:thin:@[host]:[port]:[sid], for example: `jdbc:oracle:thin:@mydb.mycompany.com:1521:orcl`
WEB_bipub.reportFolder	The chosen environment name; such as *Test*, *Training*, or *Production*.
WEB_bipub.JDBCUserName	The user name for the data source as defined in the BI Publisher data source definition page.
WEB_bipub.JDBCPassword	The password for the data source as defined in the BI Publisher data source definition page.

Note: *`dataSourceName`* parameter takes precedence over the JDBC configuration; if both are configured, the *`dataSourceName`* will only be used. Also, if no parameters are specified, then the report data source will revert back to the default data source set for the data model associated with the report.

Installation Steps

1. Log into BI Publisher (`http://<BIP server name>:9704/xmlpserver/`) as the Administrator from a browser that has access to the WebSwitching folder. The files it contains can be copied from its default installation directory to a PC of your choice.
2. Set up a database connection by going to the Oracle BI Publisher Administration page, navigating to the **JDBC Connection** page under the Data Sources section, and then click **Add Data Source**.
3. In the **Name** field, enter `<reportFolder>` (*i.e.*, the value of the `WEB_bipub.reportFolder` parameter).
4. Set the **Driver Type** to Oracle 9i/10g/11g.
5. Set the **Database Driver Class** to `oracle.jdbc.OracleDriver`.
6. Set the **Connection** string to:
   `jdbc:oracle:thin:@<your machine>:1521:<the ORACLE_SID>`
7. Set the **username** and **password** to match the your Oracle Utilities Network Management System database login values.
8. Click **Test Connection** and verify that it is properly configured.
9. Click **Apply.**
10. From the **BI Publisher Catalog** page, select **Shared Folders** from the folders tree.
11. On top of the folders section, click the **New** drop down and select **Folder** from the list.
12. In the Name field, enter `<reportFolder>`. Click **Create**. The new `<reportFolder>` folder will be added to the page.
13. Expand the `<reportFolder>` folder.
14. On top of the folders section, click the **New** drop down and select **Folder** from the list.
15. Enter `Resources` as the folder name, then click **Create**. The new folder is added.
16. Expand the **Resources** folder.
17. In the task section on the bottom left of the page, click **Upload**.
18. Browse to the `WebSwitching/Resources` report package folder and upload the archive WebSwitching.xsbz, which contains the sub-template file.

    Note: when uploading, select the “Overwrite existing report” option followed by the **Upload** button.
19. Navigate back to `Shared Folder/<reportFolder>/Resources` in the catalog page. Upload the following files from the WebSwitching report package:
    - WebSwitching.xdmz
    - WebSwitchingMiscLog.xdmz
    - PlannedSheet.xdoz
    - OutageCorrectionSheet.xdoz
    - TemplateSheet.xdoz
    - StandaloneSafety.xdoz
    - MiscLog.xdoz
20. Edit newly uploaded WebSwitching.xdmz and WebSwitchingMiscLog.xdmz and change ‘Default Data Source’ to the `<reportFolder>`.

21. Edit OutageCorrectionSheet.xdo. From the toolbar, change the data model to the newly modified WebSwitching.xdm.

    By default, the main template imports the sub-template from “Shared Folder/WebSwitching/WebSwitching.xsb. If it does not exist or you wish to change it to the newly updated sub-template, edit the report and click **Edit** (below the OutageCorrectionSheet layout) to change the import path. Select the Help option from the menu for more in-depth information on editing templates.

22. Repeat step 8 for the remaining reports, except for the MiscLog, which should be set to the WebSwitchingMiscLog data model.

23. Restart the WebLogic, DBService and JMService.

You should now be able to preview and print reports from the Web Switching application.

Switching Sheet Email Attachment Configuration

Web Switching Management’s default behavior when emailing a switching sheet is to automatically name the switching sheet and attach the file to a new email message. The default naming convention is:

- {*sheet type*}_{*sheet index*}.{*report format*}

  Examples:

  - Planned_1003.pdf
  - Emergency_1004.rtf

An alternative configuration options allows you to provide the user with a **Save As** dialog to name the switching sheet prior to attachment to the message. To implement this option, you will need to change the value of the `$SKIP_SAVE_DIALOG$` parameter from *`true`* to *`false`* in any of the files that you wish to turn the **Save As** dialog on for:

- SwmanPlannedTool.xml
- SwmanEmergencyTool.xml
- SwmanOutageCorrectionTool.xml
- SwmanTemplateTool.xml
- SwmanMenuBar.xml
- SwmanToolBar.xml
- SwmanStandAloneSafetyTool.xml

To force the **Save As** dialog for all the sheet types and for stand alone safety documents, set the `$SKIP_SAVE_DIALOG$` parameter from true to false in the following files:

- SwmanToolBar.xml
- SwmanStandAloneSafetyMenuToolBar.xml

Altering and/or Translating the Web Switching Reports

Adding XLIFF translation file

The Web Switching reports used for printing can be easily translated to alternate languages or the labels updated to something more appropriate to the project. Simply edit the report layout, open the 'Layout Properties' page and click **Extract Translation**. Within this XML file you will find a number of `<trans-unit>` elements with `<source>` and `<target>` sub-elements. Update the `<target>` entry with your translated or altered label.

For all languages except Chinese and Portuguese (Brazil), if you wish to create a language specific version of the XLIFF file, name the translated report file according to the following standard:

```
WebSwitching_<language_code>.xlf
```

where *<language_code>* is the two-letter ISO language code (in lower case). For example, `WebSwitching_en.xlf for English.`

For Chinese (China), Chinese (Taiwan), and Portuguese (Brazil) you must use the language code and territory code in the translated file name as follows:

```
WebSwitching_zh_CN.xlf
WebSwitching_zh_TW.xlf
WebSwitching_pt_BR.xlf
```

For more information on translating reports, see the section "Translating Reports" in the *Oracle Business Intelligence Publisher User's Guide.*

In order to utilize a language specific XLIFF file, the WEB_bipub.locale parameter has to be set correctly in the CES_PARAMETERS table. For example, if the language is English, the XLIFF file name would be **`WebSwitching_en.xlf`** and **`WEB_bipub.locale`** would be set to **`en-US`** in the CES_PARAMETERS table.

Updating the Sub-Template and Template files

The sub-template and template files can be altered to accommodate project requirements.

From the **BI Publisher Reports** tab, install the Template Builder by selecting and executing the downloaded executable. Once installed, use Microsoft Word to edit the sub-template and template files. Labels and the layout of data entries can be easily manipulated from this editor.

A new menu named **Oracle BI Publisher** will be added to Microsoft Word. Select the **Help** option from menu for more in-depth information on editing templates.

Updating the Report File

The templates uses data extracted from queries defined in the BI Publisher data model. For Web Switching, the report file is named WebSwitching.xdm; for the Miscellaneous Log, the report file is named WebSwitchingMiscLog.xdm.The report files can be found in the WebSwitching folder. The files contain a number of queries that are used to gather the data displayed in the reports. The Microsoft Word BIP Template Editor utilizes this schema file to assist you in adding elements to the templates. The WebSwitching.xdm data model should only be altered if additional data is required in the print or print preview report. The WebSwitchingMiscLog.xdm data model can also be altered if additional data is required in the Miscellaneous Log print report.

Changing Date Formats

The BI Publisher report templates format dates using the `NMS_DATE_FORMAT`. The `NMS_DATE_FORMAT` is based on the value of Centricity.DateTimeFormat, which is a parameter in the CentricityTool.properties file. To change the date format in your reports, edit CentricityTool.properties, which is found in the src/config/product/jconfig/global/properties directory. The Miscellaneous Log report only uses dates without times. To alter the date format for the Miscellaneous Log, update the `DBDateFormat` parameter in the CentricityTool.properties file.

Contents of the WebSwitching Folder

- oracle_sig_logo.gif
  - Oracle logo used in the header of the generated report.

- WebSwitching.xdmz
  - WebSwitching.xdm: BI Publisher data model. This file defines the data that is used by all the switching sheet and safety document reports. It contains all the queries that are used to pull the data from the database. This includes Web Switching, Event, Crew, Customer and Web Safety information.

- WebSwitchingMiscLog.xdmz
  - WebSwitchingMiscLog.xdm: BI Publisher data model. This file defines the data that is used to populate the Miscellaneous Log report. It contains all the queries that are used to pull the data from the database.

- WebSwitching.xsbz
  - WebSwitching.xsb: BI Publisher sub-template. This file is consists of template definitions for all the common section of a Web Switching report that are called from all the Web Switching report templates.

- PlannedSheet.xdoz
  - _report.xdo: BI Publisher report file. This file defines the data model, layout, properties and the translations available for Planned and Emergency Sheet report.
  - PlannedSheet.rtf: BI Publisher report template file. This file includes the PlannedSheet-specific layout of the data within the report.

- MiscLog.xdoz
  - _report.xdo: BI Publisher report file. This file defines the data model, layout, properties and the translations available for the Miscellaneous Log report.
  - MiscLog_en_US.rtf: BI Publisher report template file. This file includes the MiscLog-specific layout of the data within the report.

- MiscLog.xlf
  - Translation file used to alter the labels in the Miscellaneous Log report without having to change the report template.

- OutageCorrectionSheet.xdoz
  - _report.xdo: BI Publisher report file. This file defines the data model, layout, properties and the translations available for the Outage Correction Sheet report.
  - OutageCorrectionSheet.rtf: BI Publisher report template file. This file includes the OutageCorrectionSheet-specific layout of the data within the report.

- TemplateSheet.xdoz
  - _report.xdo: BI Publisher report file. This file defines the data model, layout, properties and the translations available for the Template Sheet report.
  - TemplateSheet.rtf: BI Publisher report template file. This file includes the TemplateSheet-specific layout of the data within the report.

- StandaloneSafety.xdoz
  - _report.xdo: BI Publisher report file. This file defines the data model, layout, properties and the translations available for the Stand Alone Safety report.
  - StandaloneSafety.rtf: BI Publisher report template file. This file includes the StandaloneSafety-specific layout of the data within the report.

Chapter 18

Building Custom Applications

The intended audience for this chapter are software programmers responsible for building interfaces and applications that interact with the Oracle Utilities Network Management System. This chapter includes the following topics:

- **Overview**
- **Prerequisites**
- **Compiling C++ Code Using the Software Development Kit**

Overview

This chapter describes how to build C++ and Java applications that interact with the Oracle Utilities Network Management System using the Oracle Utilities Network Management System Software Development Kit (SDK).

Most Oracle Utilities Network Management System implementations will require at least one custom built application, a model interface, while other implementations may have addition interfaces and other programs that interact with the Oracle Utilities Network Management System. To support the implementation of these interfaces and programs, the Oracle Utilities Network Management System has provided a Software Development Kit. The Software Development Kit is installed into the $CES_HOME/build directory and is pointed to using the .nmsrc environment variable $NMS_BUILD.

There are two subcomponents to the Software Development Kit:

$NMS_BUILD/make	The make rules to support the architecture and platform configuration.
$NMS_BUILD/include	The C++ header files required to interact with the Oracle Utilities Network Management System.
$CES_HOME/sdk/java/lib	The jar files containing compiled Java classes required to interact with the Oracle Utilities Network Management System.
$CES_HOME/sdk/java/docs	Documentation for the Oracle Utilities Network Management System Java API.
$CES_HOME/sdk/java/samples	Sample Java applications. In this release, a sample MultiSpeak-based AMR or AVL adapter is included.

Note the following regarding usage of the Oracle Utilities Network Management System Software Development Kit:

- The SDK interfaces are not documented and are for use as-is.
- The SDK interfaces may change from release to release with no guarantees of forward or backward compatibility.
- The use of the SDK can impact the running Oracle Utilities Network Management System based on what is programmed with the SDK. Impacts may include performance issues, system lock ups, system instability, data loss, and changes to system functionality. It is recommend that you heavily test any interfaces or programs you create and judge the impact on the Oracle Utilities Network Management System and understand these interfaces and programs should be considered “use at your own risk”.
- The SDK may not be used to reverse engineer the features and functionality of the Oracle Utilities Network Management System.

Prerequisites

In addition to the prerequisites required to run the Oracle Utilities Network Management System, the following are required to use the Oracle Utilities Network Management System Software Development Kit:

- GNU Make
- Apache Ant
- JDK
- Java EE 6 SDK

See the Oracle Utilities Network Management System Quick Install Guide for version information.

Verify that your .nmsrc was generated using the template from $CES_HOME/templates/nmsrc.template and that the environment variable $NMS_BUILD is set to $CES_HOME/build.

Compiling C++ Code Using the Software Development Kit

Place the C++ source code to build the custom interface or program in a subdirectory of the $NMS_CONFIG directory, typically $NMS_CONFIG/apps. The executables resulting from the compile will be generated into the $NMS_CONFIG/bin directory via the Makefile so the nms-install-config process can copy them to the runtime directory, $NMS_HOME/bin. If you create custom shared libraries, these need to be copied into $NMS_CONFIG/lib so they also are available for nms-install-config to copy them to the runtime directory, $NMS_HOME/lib.

The following is an example Makefile for the $NMS_CONFIG/apps directory:

```
######################################################################
#
# Example $NMS_CONFIG/apps directory Makefile
#
######################################################################
# Include compiler and architecture dependent Makefile parameters.
HAS_GUI = YES
include $(NMS_BUILD)/make/make.rules
LOCALLIBS = $(PP_LIB) $(MV_LIB) $(SUPPORT_LIBS) $(MB_LIB) $(GRWINDOW_LI

# Source for all run-time applications
SOURCES = \
        CustomInterface.C
OBJECTS = $(SOURCES:.C=.$(OBJ_EXT))
PROGRAM = CustomInterface$(EXE_EXT)

######################################################################
# Targets

include $(SIMPLE_PROGRAM_MAKE)

all:: $(PROGRAM)
        @ if [ ! -d "$(NMS_CONFIG)/bin" ]; then \
            mkdir $(NMS_CONFIG)/bin; \
          fi
        cp $(PROGRAM) $(NMS_CONFIG)/bin;
```

The target executable file in this example is `CustomInterface` and the C++ source code to compile is `CustomInterface.C`.

From the command prompt within the $NMS_CONFIG/apps directory, build the custom program with “make clean” to remove old compiled binaries and “make” to compile and install the binaries into the $NMS_CONFIG/bin directory.

Below is an example of what the output from the make system will look like as a result or running these two commands.

```
nms-vm:nms1> cd ~/OPAL/apps
nms-vm:nms1> make clean
rm -f *.o *~ core .pure* gmon.out  so_locations *.sl *.so *.a
rm -f \#* 3log *.third *.third.*
rm -rf ptrepository cxx_repository Templates.DB SunWS_cache tempinc
rm -f OPAL_preprocessor
nms-vm:nms1> ls
Makefile  OPAL_imp_exp.C  OPAL_preprocessor.C  OPAL_preprocessor.h
nms-vm:nms1> make onsite
OPAL_preprocessor.o
g++ -pedantic -W -Wall -Wno-format-y2k -Woverloaded-virtual -Wpointer-arith -Wca
st-align -Wwrite-strings -Wno-long-long -Wsign-promo  -g   -DDIFFUSION_NOTIFIES
   -DLINUX -D_REENTRANT -DP_THREADS   -DHAS_XT -DFAULT_RESTORATION -DGSOAP_VERSI
ON= -I/users/nms1/nms/product/1.10.0.0/build/include   -I/users/nms1/nms/product
/1.10.0.0/isis/include  -I/opt/oms-10.1/include     -c  OPAL_preprocessor.C
motif Building OPAL_preprocessor:
g++ -pedantic -W -Wall -Wno-format-y2k -Woverloaded-virtual -Wpointer-arith -Wca
st-align -Wwrite-strings -Wno-long-long -Wsign-promo  -g   -DDIFFUSION_NOTIFIES
   -DLINUX -D_REENTRANT -DP_THREADS   -DHAS_XT -DFAULT_RESTORATION -DGSOAP_VERSI
ON= -I/users/nms1/nms/product/1.10.0.0/build/include   -I/users/nms1/nms/product
/1.10.0.0/isis/include  -I/opt/oms-10.1/include     -L/users/nms1/nms/product/1.
10.0.0/lib  -o OPAL_preprocessor OPAL_preprocessor.o  -lPp  -lMv  -lMv  -lApp -L
/opt/oms-10.1/lib -lxrttable -lpdsutil -lxrttablestub -L/opt/oms-10.1/lib -lXpm
 -lCrew  -lPp  -lService  -lMB  -lGrWindow   -lintersys_xt  -lWrapper  -lBase  -
lfoss -L/users/nms1/nms/product/1.10.0.0/lib -L/users/nms1/nms/product/1.10.0.0/
isis/lib -lisisX -lisis -lisis_task_native  -lCmdLine  -L/opt/oms-10.1/lib -lMrm
 -lXm -lXp -lXext  -L/opt/oms-10.1/lib -lXt -lX11  -lpthread -ldl  -L/opt/oms-10
.1/lib  -lgsoap++  -lgsoap
Building and Linking C++ OPAL_imp_exp:
g++ -pedantic -W -Wall -Wno-format-y2k -Woverloaded-virtual -Wpointer-arith -Wca
st-align -Wwrite-strings -Wno-long-long -Wsign-promo  -g   -DDIFFUSION_NOTIFIES
   -DLINUX -D_REENTRANT -DP_THREADS   -DHAS_XT -DFAULT_RESTORATION -DGSOAP_VERSI
ON= -I/users/nms1/nms/product/1.10.0.0/build/include   -I/users/nms1/nms/product
/1.10.0.0/isis/include  -I/opt/oms-10.1/include     -L/users/nms1/nms/product/1.
10.0.0/lib   -o OPAL_imp_exp OPAL_imp_exp.C  -lPp  -lMv  -lMv  -lApp -L/opt/oms-
10.1/lib -lxrttable -lpdsutil -lxrttablestub -L/opt/oms-10.1/lib -lXpm  -lCrew
-lPp  -lService  -lMB  -lGrWindow   -lintersys_xt  -lWrapper  -lBase  -lfoss -L/
users/nms1/nms/product/1.10.0.0/lib -L/users/nms1/nms/product/1.10.0.0/isis/lib
-lisisX -lisis -lisis_task_native  -lCmdLine  -L/opt/oms-10.1/lib -lMrm -lXm -lX
p -lXext  -L/opt/oms-10.1/lib -lXt -lX11  -lpthread -ldl  -L/opt/oms-10.1/lib  -
lgsoap++  -lgsoap
cp OPAL_preprocessor OPAL_imp_exp ../bin
nms-vm:nms1> []
```

Note: By default, project compiles produce debug builds. To improve performance, you can change to optimized mode by adding the following to the.profile configuration file in your compilation environment:

```
export NMS_COMPILE_OPTIMIZED=1
```

After you have successfully compiled the custom application, run `nms-install-config` to pick up the executables from the $NMS_CONFIG/bin and install them into $NMS_HOME/bin.

Building Sample AMR and AVL Adapter

The sample AMR/AVL adapter is a J2EE application, which is intended to be deployed on J2EE application server (for example, Oracle WebLogic Server).

Required Software

JDK 1.6 Update 20 or later

Java EE 6 SDK or WebLogic application server (10.3.5).

Oracle Utilites Network Management System SDK

Build Instructions

Source code for the sample adapter is located in the `$CES_HOME/sdk/java/samples/amr` directory.

1. Edit the build.properties file:

   a. Specify the location of the NMS SDK.

      The NMS SDK is expected to be in the $CES_HOME/sdk directory. The *nms.sdk.dir* property can be used to specify different location.

      Uncomment the nms.sdk.dir property line and add the appropriate path.

      ```
      # Location of NMS SDK.
      # If not set then CES_HOME environment variable is used
      # to locate it.
      #
      nms.sdk.dir = path_to_the_dir
      ```

   b. Specify location of the Java EE jar files.

      - If using WebLogic application server and the MW_HOME environment variable is set, then the jar files from $MW_HOME/modules directory will be used.
      - If Java EE 6 SDK is used, then you will have to set the *javaee6.sdk.dir* parameter to point to the Glassfish installation.

        ```
        #
        # If using Java EE 6 SDK to build the adapter this property
        # should be set to the Glassfish install directory.
        #
        javaee6.sdk.dir = path_to_the_dir
        ```

3. From a command prompt, execute the following command:

   ```
   ant clean all
   ```

If the build is successful, the build file, demo.ear, will be created in the build directory.

Deployment

In order to run the sample application, the Oracle Thin JDBC driver and Apache log4j package have to be available on the application server where the adapter will be deployed.

The sample adapter uses subset of the configuration options of the Oracle Utilities Network Management MultiSpeak Adapter. See Chapter 10 (MultiSpeak Adapter) of the *Oracle Utilities Network Management Adapters Guide* for configuration and deployment instructions.