Oracle Utilities Network Management System

Adapters Guide

Release 1.10.0.5.0

E35153-01

May 2012

ORACLE

Oracle Utilities Network Management System Adapters Guide, Release 1.10.0.5.0

E35153-01

Contents

Preface **1-xi**
Audience 1-xi
Related Documents 1-xi
Conventions 1-xi
Common Terminology 1-xii

Chapter 1

Generic IVR Adapter **1-1**
Introduction 1-1
Supported Application Data Flows 1-1
IVR Data Flows with Oracle Utilities Network Management System 1-1
CIS's Data Flows with Oracle Utilities Network Management System 1-2
Callbacks Application Data Flows with Oracle Utilities Network Management System 1-2
Interaction Diagram 1-3
Data Flow Details 1-3
Overview 1-3
Trouble Calls 1-4
Event Status 1-5
Affected Customers 1-5
Callback Requests 1-6
Callback Requests Notes 1-7
Callback Responses 1-7
Adapter Installation 1-9
Ensure that the Generic IVR Adapter is installed. 1-9
Setup the Generic IVR Adapter System Variables 1-9
Configure Adapter to run as NMS System Service 1-9
IVRAdapter Command Line Options 1-10
Load the Generic IVR Adapter Database Tables and Stored Procedures 1-17
Software Configuration 1-18
Overview 1-18
Trouble Call Mapping Configuration 1-18
Mapping to Customer-Defined Fields in Oracle Utilities Network Management System's INCIDENTS table 1-25
Trouble Callback Mapping Configuration 1-25
SRS Rules Configuration 1-26
Map Customer-Defined Fields in the INCIDENTS table 1-26
callbackInterfaceEnabled SRS Rule 1-27
useExternalCause SRS Rule 1-27
customerPhoneParentheses SRS Rule 1-28
defaultCallbackAgent SRS Rule 1-28
callbackFeederTimeout SRS Rule 1-29
streetXsectionOffset SRS Rule 1-29
Generic IVR Adapter Trouble Call Performance 1-30
Generic IVR Adapter Troubleshooting 1-30
Database Schema 1-32

Overview 1-32
Database Table Schema 1-32
Stored Procedure Parameters 1-47
Terminology 1-61

Chapter 2

Generic SCADA Adapter 2-1
Introduction 2-1
Generic SCADA Adapter Configuration 2-1
Overview 2-1
Configure Adapter to run as NMS System Service 2-1
Command Line Options for Generic SCADA Adapter 2-2
Software Configuration 2-3
Overview 2-3
Adapter Configuration 2-3
Data Flows 2-9
SCADA Entry 2-9
Information Model 2-11
Database Schema 2-11
.... 2-16

Chapter 3

SmallWorld GIS Adapter 3-1
Model Build Process Overview 3-1
Adapter Overview 3-2
Obtaining the Software 3-2
Adapter Capabilities 3-2
Feature Extraction 3-3
Object Identifier 3-3
Connectivity 3-3
Geometric Information 3-3
Coded Value Look Up 3-3
Class Names Containing Space Characters 3-3
Oracle Utilities Network Management System Partitions with Index Maps 3-4
How to Build Smallworld Images 3-4
GE Smallworld CST 4.0 Systems 3-5
Setting up the Oracle Utilities Network Management System Smallworld Adapter 3-6
Map Tiles 3-6
Visibilities 3-6
Object Configuration 3-7
Prime Location 3-7
OMSA Object ID 3-7
OMSA_internal_associated_objects 3-7
Dataless_rwo_joins 3-7
OMSA_associated_objects 3-7
Non_joined_connected_objects 3-8
OMSA_additional_info 3-8
OMSA_extract_world 3-8
Temporary Object 3-8
OMSA_proto_record 3-8
Parameter Configuration 3-8
OMSA_component Parameters 3-9
Other Parameters 3-11
Incremental Adapter Parameters 3-11
Configuration Files 3-12
Data Model 3-13

Execution 3-13
Adapter Execution 3-13
Launching Adapter Interactively 3-13
Batch Execution 3-13
Topology 3-14
Output of Adapter 3-15
Performance 3-15
Data Model 3-15

Chapter 4

ESRI ArcGIS 9.x Adapter 4-1
Adapter Overview 4-1
Information Model 4-3
Hardware and Software Environment 4-4
Related Documents 4-5
Modeling Guidelines 4-6
Supporting Oracle Utilities Network Management System Partitions with Index Maps 4-6
Supporting Oracle Utilities Network Management System Symbology 4-6
Supporting Change Detection 4-6
ArcGIS Data Maintenance 4-8
Supporting Oracle Utilities Network Management System Data Requirements 4-10
Modeling Stations and Substations 4-14
Modeling Spatially Dense Objects 4-15
Modeling Complex Objects 4-15
Modeling Implied Objects 4-15
Mapping GIS Classes to Oracle Utilities Network Management System Classes 4-15
Adapter Functionality 4-18
Adapter Capabilities 4-18
Installing ArcGIS to Oracle Utilities Network Management System Adapter 4-23
System Configuration 4-23
Installing the Application 4-24
Obtaining the Software 4-24
Automated Installation 4-24
Application Reference Database File 4-24
Testing the Installation 4-25
Configuring the Adapter 4-25
Reference Database Tables 4-25
Explicitly Specifying Data Ownership 4-38
Freeze Versions Created and Used by the Adapter 4-38
Freeze Naming Conventions 4-39
Managing Freezes 4-39
Running the Adapter 4-39
Launching Adapter with Command Line Parameters 4-39
Launching Adapter Interactively 4-40
Populate the Required Relationship Tables 4-42
Usage Scenarios 4-43
Manually Selected Partitions (no change detection) 4-43
Save and Recall a Partition Queue 4-43
Perform Change Detection Via the Adapter 4-44
Manually Select Partitions and Save Selection to a Partition Set 4-44
Load Partition Queue from Externally Generated Source 4-45
Bypass Use of the ESRI ArcSDE Database Versions 4-45
Populating the Partition Listbox Using PartitionQueue Table 4-45
Read Predefined Partition Queue from PartitionQueue Table 4-46
Partition Loading 4-46

Load All Partitions 4-46
Load Externally Generated Partition Queue Saved in PartitionQueue 4-46
Output of Adapter 4-47
Model Preprocessor File 4-47
Extract Log File 4-48
Work Order Time Stamp File 4-49
Change Detect Log File 4-49
Performance 4-51
Issues 4-51
Windows Resources 4-51

Chapter 5

Intergraph G/Electric Adapter 5-1
Introduction 5-1
Adapter Overview 5-2
Adapter Capabilities 5-3
Partition Details 5-3
Feature Extraction 5-3
Object Identifier 5-4
Connectivity 5-4
Geometric Information 5-4
Coded Value Look Up 5-4
Names Containing Space Characters 5-4
Node Reduction 5-4
Partitions with Index Maps 5-4
Programs and Files 5-5
G/Electric Data Adapter Installation Phases 5-5
Minimum System Requirements 5-5
Disk Space Requirements 5-6
Installing the G/Electric Data Adapter 5-6
Obtaining the Software 5-6
Application Installation 5-6
Executing the Adapter Installer 5-6
Adapter Configuration 5-8
Defaults Table 5-9
Partition_Reference Table 5-11
ExcludeFeature Table 5-14
ExcludeComponent Table 5-15
PartitionGroupFeature Table 5-15
V_SPL_FIDFNO View 5-16
Third Party Products 5-16
Oracle Client Configuration 5-17
Executing the Adapter 5-17
Launching the Adapter Interactively 5-18
Launching the Adapter in Batch Mode 5-20
Output of Adapter 5-21
Performance 5-21
Application Process Flows 5-22
GIS Data Requirements 5-24
Required Data for All Adapter Features 5-24
Optional Data for All Adapter Features 5-24
Required Attributes for Electrical Objects 5-25
Optional Attributes for Electrical Objects 5-25
Sample Model Preprocessor File 5-25
Uninstalling the G/Electric Data Adapter 5-27

Microsoft Windows 2000 5-27
Microsoft Windows XP 5-27

Chapter 6

Generic WebSphere MQ Adapter 6-1
Introduction 6-1
Terminology 6-2
Hardware and Software Requirements 6-2
Oracle Utilities Network Management System Environment 6-2
External System Environment 6-3
Required Installed Software: 6-3
Functional Description 6-4
Context Diagram 6-4
Adapter Installation 6-5
Overview 6-5
Check if the Generic WebSphere MQ Adapter is installed 6-5
Configure Adapter to run as NMS System Service 6-5
Stopping the original default Q manager listener 6-10
High Availability 6-12
Clustering 6-12
Non-Redundant Queue Approach 6-12
Synchronization Process 6-12
Troubleshooting 6-12
Performance 6-13
Functional Requirements 6-13
Design Overview 6-14
Data Flows 6-14
Overview 6-14
Create Incident 6-15
Get Customer Outage Status 6-15
Get Customer Outage History 6-15
Create, Delete, Update, Get Condition 6-15
Outage Status 6-15
Create, Delete, Update, Get Customer 6-16
SQL Transactions 6-16
SQL Query 6-16
Status Check 6-17
Errors 6-17
Customer Disconnect / Reconnect. 6-17
Crew Outage Status Changes 6-17
Sending Crew Updates / Getting Crew (Request / Reply) Information 6-19
Published Crew Information Updates to an External System 6-19
Network Trace Includes Planned Outage Request and Current Feeder Request 6-19
Area Summary 6-20
Callback List 6-20
Information Model 6-20
SRS Output and SRS Output Status Message Tags 6-21
Customer Message Tags 6-29
Trouble Call Message Tags 6-29
Crew Message Tags 6-31
Crew Outage Status Changes 6-32
Configure Queues for Required Data Flows 6-32
Table Used to Define MQ Queues 6-34
XSL Transformation Files 6-34
Default CES_GET and CES_PUT queues 6-34

Trigger of broadcasting messages 6-34
.... 6-35

Chapter 7

Generic WebSphere MQ Mobile Adapter 7-1
Introduction 7-1
Overview Description 7-1
Terms 7-2
Functional Description 7-3
Functional Requirements 7-3
Hardware and Software Requirements 7-4
External System Environment 7-5
Required Installed Software 7-5
Adapter Installation 7-5
Overview 7-5
Configure Queues for Required Data Flows 7-10
Design Overview 7-11
Configuration Concepts 7-11
Output Documents 7-12
Input Documents 7-12
Integration with System Services 7-12
Aggregation of Objects 7-13
Information Flows 7-13
High Availability 7-13
Performance 7-14
High Level Messages 7-14
Information Models 7-15
Configuration 7-15
DML Files 7-16
Operator Summary 7-19
Input Elements and Attributes 7-20
SQL Insert and Update Statements 7-21
If Statement 7-22
Flags 7-23
Defaults 7-24
Alternate Names 7-24
Change Detection 7-26
The External Objects 7-27
Input Element Arrays 7-28
For Statement 7-30
Queue Specification 7-31
Association of an Output Document to the Order Object 7-31
Triggers 7-31
The Root Element 7-34
The Base Path 7-35
Stop Statement 7-36
Include Statement 7-36
Configuration Documents 7-37
Pseudo Relationships 7-40
Configuration Tables 7-40
The Order Tables 7-40
The Relationship Tables 7-44
The Code Mapping Tables 7-46
The SRS Message Type Table 7-47
The High Priority Category Table 7-48

Run Time Errors 7-49
DML Examples 7-49
An Output Document 7-49
An Input Document 7-52
DML Reference 7-56
Lexical Conventions 7-56
Basic Concepts 7-57
Type(s) 7-57
Definitions and References 7-57
Attributes 7-59
Entity Modifiers 7-60
External Data 7-60
Functions 7-61
Expressions 7-61
Lists 7-62
Statements 7-62
Documents 7-65
Configuration Document 7-66
Order of Document Processing and Other Considerations 7-67
Ordering of Incidents in the Incident Object 7-68
DML Function Calls 7-69
Event Object Fields 7-123
Incident Object Fields 7-127
Permanent Order Object Fields 7-129
Permanent Relationship Object Fields 7-129

Chapter 8

ICCP Adapter 8-1
ICCP Adapter Overview 8-1
ICCP Adapter Configuration 8-2
Configuring the Adapter to Run as a System Service 8-3
Populating the NMS Measurements Tables 8-9
Information Model - Database Schema 8-11

Chapter 9

MultiSpeak Adapter 9-1
Installation 9-1
Installation Overview 9-1
Adapter Installation Instructions for Oracle WebLogic Server 9-2
Adapter Installation Instructions for JBoss Application Server 9-2
Software Configuration 9-3
Changing Configuration Files 9-3
AMRInterface.properties file 9-3
AVLInterface.properties file 9-5
Credentials Files 9-6
Oracle Utilities Network Management System AMR Configuration Rules 9-6
Adapter Interface Communication Overview 9-8
Adapter Design 9-8
Supported Data Flows 9-8
AMR Business Processes 9-9
Database Schema 9-13
Entity Relationship Diagram 9-13
AMR_REQUESTS 9-13
AMR_RESPONSES 9-14
AMR_CU_METERS 9-14
AMR_REQUEST_RESP 9-15

Preface

Please read through this guide thoroughly before beginning an installation or configuration of any supported adapters for the Oracle Utilities Network Management System.

Audience

This document is intended for the administrator and the engineers responsible for installing and configuring Oracle Utilities Network Management System adapters.

Related Documents

- Oracle Utilities Network Management System Installation Guide
- Oracle Utilities Network Management System Configuration 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.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Common Terminology

The following terms and acronyms are used in one or more adapter descriptions.

Term	Definition
CIS	Customer Information System.
DMS	Distribution Management System.
DTD	Document type definition, used to define XML documents. DTDs were a precursor to XML schemas.
Generic IVR Adapter	A Unix application that generally executes on the *OMS* server machine. It supports the Trouble Call, Event Status, Affected Customers, Callback Request and Callback Response Data Flows.
HA	High Availability, where Oracle Utilities Network Management System is configured with a pair of redundant servers. This is usually in the form of a hardware cluster and a shared drive that contains the database.
Isis	Isis is a real-time implementation of message oriented middleware and comprises the **NMS** backbone, providing access to the server for each client and the communication required between tools and services.
IVR	Interactive Voice Response system.
JMService	Job Management Service. The Oracle Utilities Network Management System call processing and outage prediction engine.
MQSeries	A queue-based messaging system developed by IBM. This system has been renamed to WebSphere MQ.
NMS	Oracle Utilities Network Management System.
ODService	Object Directory Service. ODService improves performance of the NMS by caching large amounts of device information that is likely to be requested by applications. This caching allows the requests to be handled very quickly without directly accessing the database.
OMS	Outage Management System.
SCADA	Supervisory Control and Data Acquisition system.
SMService	System Monitor Service. SMService monitors the core processes in the system, essentially the services and interfaces.
XML	XML (Extensible Markup Language), is a subset of the SGML (Standard Generalized Markup Language; ISO 8879) standard.
XML Schema	An XML standard for defining XML documents.
XSL	XML Stylesheet Language Family; a family of recommendations for defining XML document transformations. Includes XSLT, XSL-FO, and XPath.

Chapter 1

Generic IVR Adapter

This chapter includes the following topics:

- **Introduction**
- **Supported Application Data Flows**
- **Interaction Diagram**
- **Data Flow Details**
- **Adapter Installation**
- **Software Configuration**
- **SRS Rules Configuration**
- **Database Schema**
- **Terminology**

Introduction

The purpose of this document is to provide an administration guide for the Oracle Utilities Network Management System Generic IVR Adapter. This document will discuss the required process for installing and configuring the Oracle Utilities Network Management System Generic IVR Adapter to run with various applications. This adapter has the following attributes:

- It is one of the adapters and tools that Oracle offers for integration with other product suites. It is a Unix application that generally executes on the Oracle Utilities Network Management System services server and is monitored through SMService.
- It has the ability to accept trouble calls from an external application and provide that external application with updates about existing outages.
- It can submit callback requests to an external application and receive callback responses from the external application.
- It can communicate with several external applications, such as Interactive Voice Response (IVR) systems, Customer Information System (CIS) and Callback applications.

Supported Application Data Flows

IVR Data Flows with Oracle Utilities Network Management System

The following are the Data Flows between an IVR system and Oracle Utilities Network Management System using the Generic IVR Adapter

- Creation of trouble calls from the IVR system to Oracle Utilities Network Management System
- Event status information from Oracle Utilities Network Management System to the IVR system
- Affected customer information from Oracle Utilities Network Management System to the IVR system
- Callback request information from Oracle Utilities Network Management System to the IVR system
- Callback response information from the IVR system to Oracle Utilities Network Management System

CIS's Data Flows with Oracle Utilities Network Management System

The following are the Data Flows between a CIS and Oracle Utilities Network Management System using the Generic IVR Adapter

- Creation of trouble calls from the CIS application to Oracle Utilities Network Management System
- Event status information from Oracle Utilities Network Management System to the CIS application

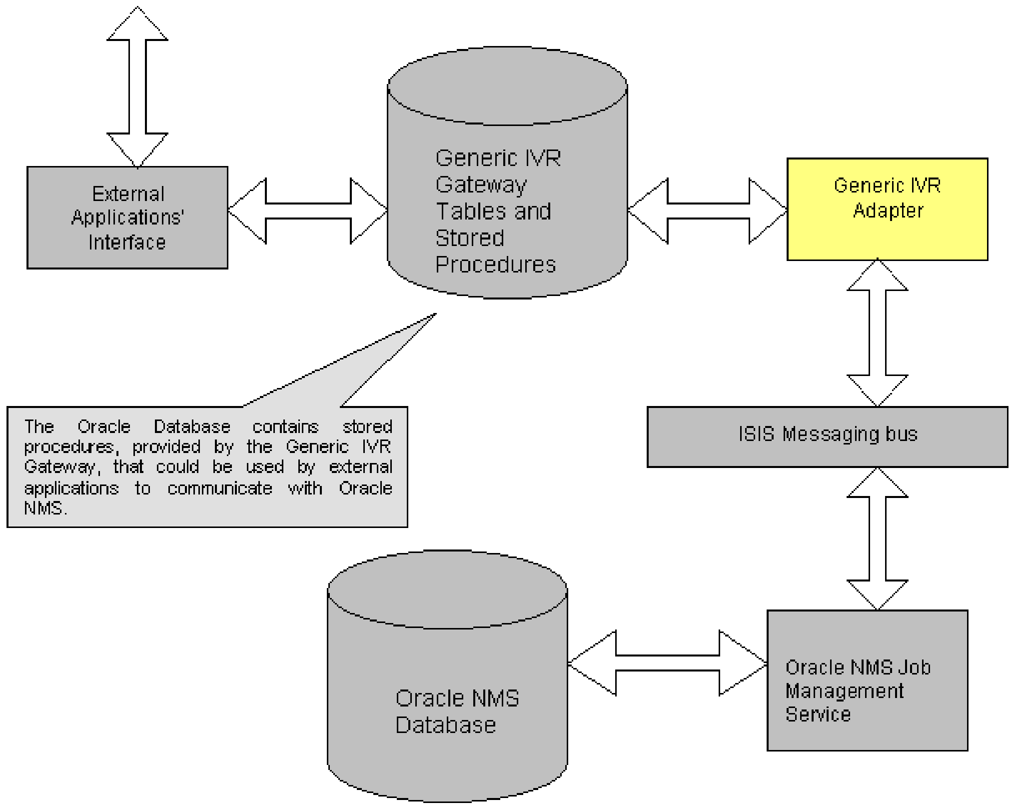
Callbacks Application Data Flows with Oracle Utilities Network Management System

The following are the Data Flows between a Callback application and Oracle Utilities Network Management System using the Generic IVR Adapter

- Callback request information from Oracle Utilities Network Management System to the Callback application
- Callback response information from Callback application to Oracle Utilities Network Management System

Interaction Diagram

Below is a diagram of the interaction between Oracle Utilities Network Management System and various external applications via the Generic IVR Adapter.



Note: In this document, it is assumed that the Generic IVR Adapter's tables and stored procedures would reside in the database used by Oracle Utilities Network Management System.

Data Flow Details

Overview

This section discusses in detail the data flows that are relevant to the Generic IVR Adapter. The data flows all involve bilateral database tables that are populated or polled by the adapter. The adapter data flows are turned on through command line switches, but the actual data transfer may be effected through the use of stored procedures. The Event Status and Affected Customers data flows have alternate stored procedures that access Oracle Utilities Network Management System operational tables rather than the advertised bilateral tables. The alternative procedures offer greater reliability at the expense of overall Oracle Utilities Network Management System performance. Oracle recommends the use of the alternative stored procedures, but using database replicated Oracle Utilities Network Management System tables to avoid impacting the production Oracle Utilities Network Management System. When the alternative stored procedures are used, the corresponding command line switches should NOT be turned on.

The documentation in this section assumes the use of the dedicated bilateral tables.

Trouble Calls

New trouble calls need to be sent to Oracle Utilities Network Management System to apply the outage analysis algorithm to predict the outage device. The Generic IVR Adapter provides the pr_trouble_calls stored procedure to pass trouble call information from the external application to Oracle Utilities Network Management System.

Data Flow Characteristics

The following are characteristics of the Trouble Calls Data Flow

Characteristics	Value
Table	TROUBLE_CALLS. For schema information, see **TROUBLE_CALLS Table Schema** on page 1-32.
Stored Procedure	pr_trouble_calls. For stored procedure parameter information, see **pr_trouble_calls Stored Procedure** on page 1-47.
Direction	external application to Oracle Utilities Network Management System
Generic IVR Adapter Data Retrieval Frequency to Oracle Utilities Network Management System	Periodic (configurable)

Data Flow Steps

1. The external application invokes the pr_trouble_calls stored procedure to submit a trouble call.
2. The pr_trouble_calls stored procedure inserts the trouble call in the TROUBLE_CALLS table. The CALL_STATUS field of the trouble call in the table will be set to 'N' signifying that the trouble call is new.
3. The Generic IVR Adapter polls a configurable number of new records from the TROUBLE_CALLS table within a configurable poll period. The CALL_STATUS field of the trouble call in the table is updated to 'I' (in progress) signifying that the trouble call is in the process of being submitted to the Oracle Utilities Network Management System Job Management Service (JMService).
4. Once processed, the retrieved records are submitted to Oracle Utilities Network Management System' JMService so the outage analysis algorithms could be used for the submitted trouble calls. The CALL_STATUS field of the trouble call in the table is updated to 'C' (complete) signifying the trouble call has been successfully submitted from the external application to Oracle Utilities Network Management System.

    Note: The Generic IVR Adapter also allows that some fields from the CES_CUSTOMERS table (rather than the parameter values supplied on the pr_trouble_calls stored procedure) be submitted to the Oracle Utilities Network Management System JMService.

Event Status

When an outage is created, updated or merged within Oracle Utilities Network Management System, the external application may need to know this information. The Generic IVR Adapter provides the pr_trouble_status stored procedure that the external application could use to retrieve outage information from Oracle Utilities Network Management System.

Data Flow Characteristics

The following are characteristics of the Event Status Data Flow:

Characteristics	Value
Table	TROUBLE_STATUS. For schema information, see **TROUBLE_STATUS Table Schema** on page 1-39.
Stored Procedure	pr_trouble_status. For stored procedure parameter information, see **pr_trouble_status Stored Procedure** on page 1-51.
Direction	Oracle Utilities Network Management System to external application
Generic IVR Adapter Data Update Frequency from Oracle Utilities Network Management System	Event basis

Data Flow Steps

1. The Generic IVR Adapter inserts records to or updates records on the TROUBLE_STATUS table for each outage created, updated or merged within Oracle Utilities Network Management System.
2. The Generic IVR Adapter provides the pr_trouble_status stored procedure that queries outage information from the TROUBLE_STATUS table for a given event handle parameter.
3. The external application invokes the pr_trouble status stored procedure and supplies a handle to retrieve an outage information.

Affected Customers

Customers could be affected when an outage occurs. The external application may need to know if a customer is a part or has been a part of an outage. The Generic IVR Adapter provides the pr_affected_customers stored procedure that could be used by the external application to query if a given customer is a part or has been a part of an outage. If so, the most recent **active** outage information (or the most recent **inactive** outage information, if an active outage was not found) is returned by the same stored procedure.

Data Flow Characteristics

The following are characteristics of the Affected Customers Data Flow:

Characteristics	Value
Table	TROUBLE_AFFECTED_CUSTOMERS. For schema information, see **TROUBLE_AFFECTED_CUSTOMERS Table Schema**.
Stored Procedure	pr_affected_customers. For stored procedure parameter information, see **pr_affected_customers Stored Procedure**.
Direction	Oracle Utilities Network Management System to external application
Generic IVR Adapter Data Update Frequency from Oracle Utilities Network Management System	Event basis

Data Flow Steps

1. The Generic IVR Adapter inserts records to or updates records on the TROUBLE_STATUS table for each outage created, updated or merged within Oracle Utilities Network Management System.
2. The Generic IVR Adapter also updates the TROUBLE_AFFECTED_CUSTOMERS table for each customer affected by such outage.
3. The Generic IVR Adapter provides the pr_affected_customers stored procedure that tells if a given customer is a part or has been a part of an outage, by querying the TROUBLE_STATUS and the TROUBLE_AFFECTED_CUSTOMERS tables.
    - If the customer is part of an active outage, the most recent active outage information is returned by the pr_affected_customers stored procedure. Some customer information from the TROUBLE_AFFECTED_CUSTOMERS table is returned by the same stored procedure as well.
    - If the customer is not part of an active outage anymore, but a part of an inactive outage, the most recent inactive outage information is returned by the pr_affected_customers stored procedure. Some customer information from the TROUBLE_AFFECTED_CUSTOMERS table is returned by the same stored procedure as well.
    - If the customer is not part of an active or inactive outage, no customer or event information is returned.

    Note: The pr_affected_customers stored procedure internally invokes the pr_trouble_status stored procedure to retrieve outage information.
4. The external application invokes the pr_affected_customers stored procedure and passes a customer identifier to retrieve customer information and outage information, given the rules above.

Callback Requests

A customer may request that he/she be called back as soon as the outage that he/she reported has been restored. The Generic IVR Adapter provides the stored procedure pr_trouble_callback_requests to be used by an external application that is managing the callback process. This procedure returns a list of calls where the customer has requested a callback.

Data Flow Characteristics

The following are characteristics for the Callback Request Data Flow:

Characteristics	Value
Table	TROUBLE_CALLBACKS. For schema information, see **TROUBLE_CALLBACKS Table Schema** on page 1-42.
Stored Procedure	pr_trouble_callback_requests. For stored procedure parameter information, see **pr_trouble_callback_requests Stored Procedure** on page 1-56
Direction	Oracle Utilities Network Management System to external application
Generic IVR Adapter Data Retrieval Frequency from Oracle Utilities Network Management System	Periodic (configurable)

Data Flow Steps

1. When an outage with a corresponding callback request is restored, Oracle Utilities Network Management System builds a callback list.
2. From the list, callback requests could be assigned to callback agents or to the external application, either in a manual (using Oracle Utilities Network Management System Web Callbacks) or an automated manner (via SRS rules).
3. The Generic IVR Adapter retrieves all callback requests assigned to the external application and inserts the callback requests to the TROUBLE_CALLBACKS table. The PROCESS_STATUS field of the callback request in the table would be set to 'N' signifying that the callback request is new. The CALLBACK_DONE field of the callback request in the table would be set to 'N' signifying that the callback has not yet been done.
4. The Generic IVR Adapter provides the pr_trouble_callback_requests stored procedure, which picks new callback requests from the TROUBLE_CALLBACKS table.
5. The external application could use the pr_trouble_callback_requests stored procedure to pick new callback requests. Callback requests that were picked are marked with a PROCESS_STATUS field equal to 'I' (callback response in progress) on the TROUBLE_CALLBACKS table.

Callback Requests Notes

Once an outage event or a non-outage event is restored, callbacks are generated if the call is marked for a callback. All events have a restoration, either explicit or implicit, so any event can generate a callback. Also, in the event that the customer called multiple times, the customer will receive multiple callbacks if he requested a callback on each call. JMService gathers every call associated with an event, without filtering duplicate callers. Every call that is marked for callback will receive a callback.

Callback Responses

The external application calls the customer to confirm if power has been restored or not. The result of this call is passed from the external application to Oracle Utilities Network Management System via the pr_trouble_callback_responses stored procedure.

Data Flow Characteristics

The following are characteristics of the Callback Responses Data Flow:

Characteristics	Value
Table	TROUBLE_CALLBACKS. For schema information, see **TROUBLE_CALLBACKS Table Schema** on page 1-42.
Stored Procedure	pr_trouble_callback_responses. For stored procedure parameter information, see **pr_trouble_callback_requests Stored Procedure** on page 1-56.
Direction	external application to Oracle Utilities Network Management System
Generic IVR Adapter Data Update Frequency to Oracle Utilities Network Management System	Periodic (configurable)

Data Flow Steps

1. The external application calls the customer to confirm if power has been restored. The result of this call is passed back to Oracle Utilities Network Management System via the pr_trouble_callback_responses stored procedure.
2. The stored procedure uses the incident number and premise ID combination (or the external ID and premise ID combination if the first combination is not provided) to identify a callback record in the TROUBLE_CALLBACKS table that would be receiving a response.
3. The stored procedure updates the identified callback in the TROUBLE_CALLBACKS table by updating the following fields:
    - The callback's CALLBACK_DONE field to 'Y' signifying that the callback was already done.
    - The callback's CALLBACK_TIME field. CALLBACK_TIME field defaults to the system date if no value was provided.
    - The callback's CALLBACK_STATUS field with the appropriate callback response code.
4. The Oracle Utilities Network Management System Generic IVR Adapter queries the TROUBLE_CALLBACKS table for new callback responses received and sends this information to Oracle Utilities Network Management System.
5. In Oracle Utilities Network Management System, the callback could get completed or cancelled or a new event (with the original call information) will be created, depending on the callback response.

Callback Responses Notes

When a callback is made and no response from customer is received, a callback time will still be recorded. Any callback time that is submitted with a status is propagated, even if the status is no reply from the customer. It is understood in this case to be the last attempted callback. Also, when a nested outage is found, the new call and event are backdated to the original outage time.

> **Note:** Oracle Utilities Network Management System does not track callback history.

Adapter Installation

This section describes how to install the Oracle Utilities Network Management System Generic IVR Adapter.

Ensure that the Generic IVR Adapter is installed.

- Verify that the following files are found in their respective folders
- $CES_HOME/lib/libIVRAdapter.so
- $CES_HOME/bin/IVRAdapter
- $CES_HOME/bin/ces_ivr_gateway.ces
- $CES_HOME/sql/product_retain_ivr_interface.sql
- $CES_HOME/sql/product_ivr_interface_head.sql
- $CES_HOME/sql/product_ivr_interface_body.plb
- $CES_HOME/bin/troubleCallCreate
- $CES_HOME/bin/ivrCallPerPoll.ces
- $CES_HOME/bin/ivrPollPeriod.ces

Setup the Generic IVR Adapter System Variables

Include the following variables in the system variables definition:

Variable	Value
IVR_RDBMS_USER	same as $RDBMS_USER defined in the system
IVR_RDBMS_PASSWD	same as $RDBMS_PASSWD defined in the system
IVR_RDBMS_HOST	same as $RDBMS_HOST defined in the system
IVR_ORACLE_SID	same as $ORACLE_SID defined in the system

Note: This is setup in the .nmsrc file located in the $NMS_HOME (and configured by running config_nmsrc.pl). After the setup of the system variables, make sure that the .nmsrc is rerun or a new terminal is opened. The above setup assumes that the database where the Generic IVR Adapter tables and stored procedures would reside would be the same database used by the Oracle Utilities Network Management System environment.

Configure Adapter to run as NMS System Service

Configure the Generic IVR Adapter to run as an Oracle Utilities Network Management System service by updating the $NMS_HOME/etc/system.dat file to include the Generic IVR Adapter as a system service. 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 Generic IVR Adapter. Search for IVRAdapter 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 to the Generic IVR Adapter to properly be monitored by SMService. See the following section for details on command lines options for the Generic IVR Adapter.

IVRAdapter Command Line Options

The section below lists the possible command line options for the Generic IVR Adapter. This section also introduces a tool that randomly creates trouble calls, along with its command line options. Performance tuning and high-level diagnostic messages that could be used on the Generic IVR Adapter will be discussed in this section as well.

The Generic IVR Adapter provides various command line options that enables Data Flows and configures Data Flow behavior. The following enumerates the command line options of the Generic IVR Adapter.

```
IVRAdapter -nodaemon
           -help
           -troublecall
           -omscbreq
           -omscbresp
           -eventstatus
           -cleantable
           -synchronize
           -debug [0-2]
           -callperpoll NUMBERCALLS
           -pollperiod SECONDS
           -docustquery
           -cbreqinterval SECONDS
           -cbrespinterval SECONDS
           -cleaninterval HOURS
           -keepdbinfo DAYS
           -cbagent AGENTNAME
           -cbpassword PASSWORD
         [ -cbAny | -cbLast ]
```

This section groups the Generic IVR Adapter command line options under the context of the Data Flow or Data Flows it is associated to.

Generic IVR Adapter Generic Command Line Options

The following are the Generic IVR Adapter command line options that are independent from any Data Flow:

Option	Usage	Description
help	IVRAdapter -help	Displays the available command line options
nodaemon	IVRAdapter -nodaemon	Runs the application on the foreground
debug	IVRAdapter -debug LEVEL (where LEVEL is 0, 1 or 2)	Runs gateway in debug mode. Associated number represents the debug level range from 0 to 2.

Trouble Call Data Flow Command Line Options

The following are the Generic IVR Adapter command line options that are related to the Trouble Calls Data Flow. For more information, see **Trouble Calls** on page 1-4.

Option	Usage	Description	Depends On	Default Value
troublecall	IVRAdapter -troublecall	Enables the Trouble Calls Data Flow. **Note**: This option must be enabled for CC&B - NMS integration.		
callperpoll	IVRAdapter -callperpoll NUMBERCALLS (where NUMBERCALLS is an integer)	Specifies the number of calls processed in the TROUBLE_CALLS table per poll of information.	troublecall	100 calls per poll of information
pollperiod	IVRAdapter -pollperiod SECONDS (where SECONDS is an integer)	Specifies the interval (in seconds) between two successive polls or queries from the TROUBLE_CALLS table	troublecall	a 6 second interval between two successive polls
docustquery	IVRAdapter -docustquery	If this option is selected, not all fields in the TROUBLE_CALLS table are directly fed to JMService. Instead, some of the fields would come from the CES_CUSTOMERS table. **Note**: This option should not be used in combination with the CC&B - NMS integration.	troublecall	

Event Status and Affected Customers Data Flow Command Line Options

The following are the Generic IVR Adapter command line options that are related to the Event Status and Affected Customers Data Flows. For more information, see **Event Status** on page 1-5 and **Affected Customers** on page 1-5.

Option	Usage	Description	Depends On	Default Value
eventstatus	IVRAdapter -eventstatus	Enables the Event Status and the Affected Customers Data Flows. Retrieves data from the trouble_status table.		
synchronize	IVRAdapter -synchronize	Could be used for the Event Status and Affected Customers Data Flows. Rebuilds the TROUBLE_STATUS and TROUBLE_AFFECTED_CUSTOMERS tables with the recent information from Oracle Utilities Network Management System. This is useful when the Generic IVR Adapter was down for a long period of time.	eventstatus	

Callback Requests Data Flow Command Line Options

The following are the Generic IVR Adapter command line options that are related to the Callback Requests Data Flow. For more information, see **Callback Requests** on page 1-6.

Option	Usage	Description	Depends On	Default Value
omscbreq	IVRAdapter -omscbreq	Enables the Callback Requests Data Flow		
cbreqinterval	IVRAdapter -cbreqinterval SECONDS (where SECONDS is an integer)	Specifies the interval (in seconds) between two successive polls from the list of callback requests	omscbreq	a 5 second interval between two successive polls.
cbAny	IVRAdapter -cbAny	Callback is submitted to IVR if requested by the customer during any call.	omscbreq	
cbLast	IVRAdapter -cbLast	Callback is submitted to IVR if requested by the customer during the last call.	omscbreq	

Callback Responses Data Flow Command Line Options

The following are the Generic IVR Adapter command line options that are related to the Callback Responses Data Flow. For more information, see **Callback Responses** on page 1-7.

Option	Usage	Description	Depends On	Default Value
omscbresp	IVRAdapter - omscbresp	Enables the Callback Responses Data Flow		
cbrespinterval	IVRAdapter - cbrespinterval SECONDS (where SECONDS is an integer)	Specifies the interval (in seconds) between two successive polls from the TROUBLE_CALLBACKS table for received callback responses	omscbresp	a 5 second interval between two successive polls.

Command Line Options Used by Multiple Data Flows

The following are the Generic IVR Adapter command line options that are related to multiple Data Flows. On the 'Depends On' section, the term 'any option that enables a Data Flow' would pertain to either one of the following command line options: 'troublecall', 'eventstatus', 'omscbreq' and 'omscbresp'.

Option	Usage	Description	Depends On	Default Value
cleantable	IVRAdapter - cleantable	Could be used for any of the five Data Flows. A flag that allows the Generic IVR Adapter to remove some records from its tables.	any option that enables a Data Flow	
cleaninterval	IVRAdapter - cleaninterval HOURS (where HOURS is an integer)	Could be used for any of the five Data Flows. Specifies the interval (in HOURS) between two successive attempts to delete old records from the Oracle Utilities Network Management System Generic IVR Adapter tables.	Cleantable and any option that enables a Data Flow	1 hour between to successive delete attempts
keepdbinfo	IVRAdapter - keepdbinfo DAYS (where DAYS is an integer)	Could be used for any of the five Data Flows. Records on the Generic IVR Adapter tables older than the specified number of days will be deleted. Certain criteria apply on which records of the Oracle Utilities Network Management System Generic IVR Adapter tables are removed and how the records are aged.	Cleantable and any option that enables a Data Flow	3 days

Option	Usage	Description	Depends On	Default Value
cbagent	IVRAdapter –cbagent AGENTNAME (where AGENTNAME is a string)	Could be used for the Callback Requests and Callback Responses Data Flows. The agent name that the Generic IVR Adapter uses in retrieving calls from the callback list. Valid agent names are located in CES_USER and ENV_ACCESS tables. The agent name used should be an external agent, as indicated in the CES_USER table.	omscbreq or omscbresp	IVR
cbpassword	IVRAdapter –cbpassword PASSWORD (where PASSWORD is a string)	Could be used for the Callback Requests and Callback Responses Data Flows. The password of the agent name that the Generic IVR Adapter uses in retrieving calls from the callback list. Valid agent passwords are located in the CES_USER table as well.	(omscbreq or omscbresp) and cbagent	IVR

For the keepdbinfo command line options, a record that starts aging on a given day, say 9:00 pm would we considered one day old when the next day with a time of 9:00 pm is reached (and not 12:00 am, which is just 3 hours from the time the record started aging).

troubleCallCreate Tool Command Line Options

Random trouble calls could be created and passed to the Generic IVR Adapter using the troubleCallCreate tool. The troubleCallCreate tool inserts entries to the TROUBLE_CALLS table. From here, the Generic IVR Adapter (through the Trouble Calls Data Flow) could fetch the new records from this table and pass this information to Oracle Utilities Network Management System, so Oracle Utilities Network Management System could apply the outage analysis algorithm to predict the outage device.

> **Note**: It is important for the Generic IVR Adapter System Variables to be setup to run the troubleCallCreate tool. For more information, see Setup the Generic IVR Adapter System Variables.

The following are the command line options for the troubleCallCreate tool:

Option	Usage	Description	Default Value
help	troubleCallCreate –help	Displays the available command line options	
debug	troubleCallCreate -debug	Runs this tool in debug mode, defaulting the debug level to 2.	Defaults to debug level 2
totalcalls	troubleCallCreate –totalcalls NUMBEROFCALLS (where NUMBEROFCALLS is an integer)	Specifies the number of trouble calls to be created	
troublecall	troubleCallCreate –troublecall	Creates one trouble call	

troubleCallCreate tool on testing Trouble Calls Data Flow

As the troubleCallCreate tool randomly creates trouble calls, this tool could be used to test the Trouble Calls Data Flow. For more information about this Data Flow, see **Trouble Calls** on page 1-4.

The troubleCallCreate tool uses the CES_CUSTOMERS table to retrieve some customer information that would be used as entries in the TROUBLE_CALLS table. The tool always begins querying the CES_CUSTOMERS table starting from the first row, each time it is invoked.

When multiple trouble calls would be created (using the 'totalcalls' command line option), the troubleCallCreate tool would place a different permutation of trouble code bits for each trouble call in the TROUBLE_CALLS table.

After running the troubleCallCreate tool, the results could be verified using the following database tables:

- The TR OUBLE_CALLS table is populated with a new trouble call record (or with a certain number of trouble calls, assuming that the 'totalcalls' command line option was used).
- As the Generic IVR Adapter runs (using the Trouble Calls Data Flow), the INCIDENTS table is populated with new records.

> **Note:** The number of new records in the INCIDENTS table is less than or equal to the total number of new trouble calls in the TROUBLE_CALLS table,

as Oracle Utilities Network Management System outage analysis algorithms allow grouping of calls based on various criteria.

troubleCallCreate tool on testing Callback Requests Data Flow

The Callback Requests Data Flow could be tested as well using the troubleCallCreate tool, since all trouble calls generated by such tool requires callback. For more information about this Data Flow, see **Data Flow Details** on page 1-3.

- For a generated trouble call, if part of the trouble code is described to be 'Power On', no record in the TROUBLE_CALLBACKS table will be generated even if the event is restored.

Load the Generic IVR Adapter Database Tables and Stored Procedures

- The ces_ivr_gateway.ces script is responsible for loading various SQL files responsible for creating the Generic IVR Adapter tables and stored procedures. The ces_ivr_gateway.ces script could call some or all of the following scripts depending on how it was invoked:
  - product_retain_ivr_interface.sql - responsible for dropping and recreating the Generic IVR Adapter tables.
  - product_ivr_interface_head.sql - responsible for loading the Generic IVR Adapter stored procedure head.
  - product_ivr_interface_body.plb - responsible for loading the Generic IVR Adapter stored procedure body.
- To create the Generic IVR Adapter tables and stored procedure, run the following command:

  ```
  ces_ivr_gateway.ces -offline
  ```

  Note: The command above recreates the Generic IVR Adapter table by dropping and creating it, therefore wiping out the contents of the Generic IVR Adapter tables.
- To create the Generic IVR Adapter stored procedure without dropping and recreating the Generic IVR Adapter tables, run the following command:

  ```
  ces_ivr_gateway.ces
  ```

Software Configuration

This section is intended to help the user configure the Generic IVR Adapter that was installed on the previous section. This includes the default configuration used, and the modifications to the base configurations that need to be done in order to customize the adapter's behavior.

Overview

This section will discuss the how to map pieces of trouble call information sent by the external application to specific database fields within Oracle Utilities Network Management System via the Trouble Call Mapping Properties Configuration file. Moreover, this section will discuss various SRS rules that could be used for the Generic IVR Adapter.

Trouble Call Mapping Configuration

The fields of the Generic IVR Adapter's TROUBLE_CALLS table could be mapped with the fields of Oracle Utilities Network Management System' INCIDENTS and JOBS table. This is done through column matching of TROUBLE_CALLS fields with JMS Input String (JMS.h), which is the standard product column and user-defined configuration through SRS_RULES.

For more information about the Generic IVR Adapter's TROUBLE_CALLS table, see **TROUBLE_CALLS Table Schema** on page 1-32.

Mapping to the Base Fields in Oracle Utilities Network Management System Tables

The following table explains how the base fields of the INCIDENTS and the JOBS tables of Oracle Utilities Network Management System are mapped with the fields of the TROUBLE_CALLS table of the Generic IVR Adapter.

Below is a description of each column

- The JMS Input String (first column) is the standard product column found in JMS.h, which is used to create calls with the JMS::sendJMSinput() API, within the Oracle Utilities Network Management System.
- The 'Description' column (second column) describes the content of the field.
- The 'Mapping to Oracle Utilities Network Management System Tables' column (third column) identifies to what fields of the INCIDENTS table or the JOBS table a given JMS Input String is tied up to. In this column, INC.<database field name> indicates that the field name is part of the INCIDENTS table; JOBS.<database field name> indicates that the field name is part of the JOBS table.
- The 'Mapping to TROUBLE_CALLS table' column (fourth column) identifies the TROUBLE_CALLS table column the JMS Input String is currently mapped to.

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
ADDR_BUILDING	Customer building address. The building number portion of the street address of the customer.	INC.ADDR_BUILDING	ADDR_BUILDING
ADDR_CITY	Customer city. The city or city/state portion of the address of the customer.	INC.ADDR_CITY	ADDR_CITY

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
ADDR_CROSS_ STREET	Intersection cross street name. Name of the second cross street should be in ADDR_STREET field.	INC.ADDR_CROSS_STREET	ADDR_CROSS_ STREET
ADDR_STREET	Customer street address. The full street address of the customer.	INC.ADDRESS JOBS.ADDR_STREET	ADDR_STREET
ALTERNATE_PHONE	Alternative contact number. Alternate phone number for contacting the customer.	INC.ALTERNATE_PHONE	ALTERNATE_PHONE
APPT_RANGE	Appointment Range.	INC.APPT_RANGE	APPT_RANGE
APPT_TIME	Time of appointment.	INC.APPT_TIME	APPT_TIME
APPT_TYPE	Type of appointment.	INC.APPT_TYPE	APPT_TYPE
CALL_ID	Not used.		CALL_ID
CALL_TIME	Input time of call. The input time of the incident. If not provided, the current time will be used.	INC.INPUT_TIME	CALL_TIME
CALL_TYPE	Type of call.	INC.TYPE	CALL_TYPE
CALLBACK_LATE	Callback late indicator. Indicates that it is OK to call back the customer beyond a defined 'late' time. This information is only stored in Oracle Utilities Network Management System. No other action is taken by Oracle Utilities Network Management System.	INC.CALLBACK_LATE	CALLBACK_LATE
CALLBACK_REQUEST	Indicates either a callback is requested or not.	INC.CALLBACK_REQUEST	CALLBACK_REQUEST
CALLBACK_TIME	Time callback requested. Time for which callback or a follow-up call was requested.	INC.CALLBACK_TIME	CALLBACK_TIME
CANCEL_CALL	Not used.		CANCEL_CALL

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
CHECK_CUTOFF	Check cutoff customer indicator. If set to Y, check if the customer is disconnected, using the CES_DISCONNECTS table. If the customer is disconnected, the call will not be saved, an error will be returned and the VERIFY_DISCONNECTS table will be populated.		CHECK_CUTOFF
CID_ALIAS	Not used.		CID_ALIAS
CLUE	Indicates if call is clue if set to Y.	INC.CLUE	CLUE
COMBINE_PRI	Total priority of call.		COMBINE_PRI
COMMENT	Call-taker Comments. Comments provided by the customer or call-taker about the incident.	INC.OP_COMMENT	CALL_COMMENT
CUST_ADDRESS	Not used.		CUST_ADDRESS
CUST_CALL_CANCEL	Call cancel indicator.	INC.CALL_CANCEL	CUST_CALL_CANCEL
CUST_CITY	Not used.		CUST_CITY
CUST_CRITICAL	Critical customer indicator. This is added to the critical C count of the outage.	INC.CRITICAL_CUST	CUST_CRITICAL
CUST_DEVICE_ALIAS	The name of the device to which the customer is connected. This must be the alias of the device handle provided with CUST_DEVICE_CLS and CUST_DEVICE_IDX. If not provided, the service will query ODService to get this information, incurring a performance penalty in call processing.	INC.OBJECT	CUST_DEVICE_ALIAS

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
CUST_DEVICE_CLS	Customer device class. The class part of the handle for the device to which the customer is connected. If CUST_ID is provided, but the device is not, JMService will look up the customer device in the CES_CUSTOMERS table. If the provided device is a supply node, it will be put in SUPPLY_CLS & SUPPLY_IDX and the first stage device will be put in H_CLS & H_IDX.	INC.H_CLS	CUST_DEVICE_CLS
CUST_DEVICE_IDX	Customer device index. The index part of the handle for the device to which the customer is connected. See CUST_DEVICE_CLS above.	INC.H_IDX	CUST_DEVICE_IDX
CUST_DEVICE_NCG	NCG of customer device.	INC.NCG	CUST_DEVICE_NCG
CUST_DEVICE_ PARTITION	Partition of customer device.	INC.PARTITION	CUST_DEVICE_PARTI TION
CUST_FIRST_NAME	Customer first name. The first name of the customer. If CUST_FIRST_NAME and CUST_LAST_NAME are both provided, they will be appended together with a space. The concatenated customer first and last name (with a space in the middle) may not be larger than 75 characters. This may be used for the full name of the customer if CUST_LAST_NAME is omitted.	INC.CUSTOMER_NAME JOBS.CUSTOMER_NAME	CUST_FIRST_NAME
CUST_ID	Unique identifier of a customer record in NMS. See CUST_DEVICE_CLS above.	INC.CID	CUST_ID
CUST_INTERSECT_ CLS	Intersecting device class.		CUST_INTERSECT_ CLS
CUST_INTERSECT_ IDX	Intersecting device index.		CUST_INTERSECT_ IDX
CUST_INTERSECT_ NCG	Intersecting NCG.		CUST_INTERSECT_ NCG

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
CUST_INTR_X	Intersecting X coordinate. X coordinate used for intersection grouping. See **streetXsectionOffset SRS Rule** for more information.		CUST_INTR_X
CUST_INTR_Y	Intersecting Y coordinate. Y coordinate used for intersection grouping. See **streetXsectionOffset SRS Rule** for more information.		CUST_INTR_Y
CUST_KEY	Customer account number.	INC.ACCOUNT_NUM	CUST_KEY
CUST_LAST_NAME	The last name of the customer. See CUST_FIRST_NAME above.	INC.CUSTOMER_NAME JOBS.CUSTOMER_NAME	CUST_LAST_NAME
CUST_LIFE_SUPPORT	Life support customer. If set to 'Y', indicates a life support customer. This is added to the critical K count of the outage.	INC.LIFE_SUPPORT	CUST_LIFE_SUPPORT
CUST_ORDER_NUM	Customer order number. Not used in the Oracle Utilities Network Management System.	INC.ORDER_NUMBER	CUST_ORDER_NUM
CUST_PHONE	Customer phone number. The non-area code portion of the customer phone number. If both CUST_PHONE and CUST_PHONE_AREA are provided, they will be appended according to the customerPhoneParentheses SRS rule. The concatenated customer phone number and area (including parentheses) may not be larger than 32 characters. This field may be used for the full customer phone number if CUST_PHONE_AREA is omitted. See **customerPhoneParentheses SRS Rule** for more information.	INC.CUSTOMER_PHONE JOBS.CUSTOMER_PHONE	CUST_PHONE
CUST_PHONE_AREA	Customer phone area code. The area code portion of the customer phone number. See CUST_PHONE above.	INC.CUSTOMER_PHONE JOBS.CUSTOMER_PHONE	CUST_PHOHE_AREA

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
CUST_PHONE_ UPDATE	Whether to update customer phone. If set to Y, the customer phone number will be updated in the CUSTOMER_PHONE_OVERRIDE table.		CUST_PHONE_ UPDATE
CUST_PRIORITY	Customer priority. This string is used to determine the critical customer type and priority of the customer.	INC.CUSTOMER_TYPE	CUST_PRIORITY
CUST_STATUS	Condition status of call.		CUST_STATUS
CUST_TROUBLE_ CODE	Customer complaint. The customer complaint (trouble code). This is a required field and must correspond with values in the SRS_TROUBLE_CODES table.	INC.COMPLAINT	CUST_TROUBLE_ CODE
CUST_TROUBLE_ QUEUE	Customer trouble queue.	INC.TROUBLE_QUEUE JOBS.TROUBLE_QUEUE	CUST_TROUBLE_ QUEUE
DRV_INST	Driving instructions.	INC.DRV_INSTR1	DRV_INST
EXTERNAL_ID	Unique call identifier. The unique identifier for the incident.	INC.EXTERNAL_ID JOBS.EXTERNAL_ID	EXTERNAL_ID
FUZZY_NCG_CLS	Fuzzy control zone class.		FUZZY_NCG_CLS
FUZZY_NCG_IDX	Fuzzy control zone index.		FUZZY_NCG_IDX
GENERAL_AREA	General Area. Not Used in the Oracle Utilities Network Management System.	INC.GENERAL_AREA	GENERAL_AREA
GROUP_BY_NAME	Fuzzy control zone name.		GROUP_BY_NAME
GROUPABLE	If set to Y, the call is groupable.	INC.GROUPABLE	GROUPABLE
MEET_TIME	Time of customer meet. If provided, meet created will be a future meet for the given time. Otherwise, if a meet is created it will be a critical meet. MEET_TYPE must be provided to create a meet.	INC.MEET_TIME	MEET_TIME

JMS Input String	Description	Mapping to System Tables	Mapping to TROUBLE_CALLS table
MEET_TYPE	Customer meet type. If set to 1, a new meet will be created. If set to 2, an existing meet will be rescheduled. If set to 3, an existing meet will be canceled. If any other value is provided, no meet will be created. May be used in conjunction with MEET_TIME.	INC.MEET_CODE	MEET_TYPE
METER_ID	Customer meter number.	INC.METER_ID	METER_ID
POWER_UP	Power-up call. Used for power-up messages from CellNet. Used for AMR.		POWER_UP
RELATED_EVT_APP	Related event application.		RELATED_EVT_APP
RELATED_EVT_CLS	Related event class.	INC. RELATED_CLS	RELATED_EVT_CLS
RELATED_EVT_IDX	Related event index.	INC. RELATED_IDX	RELATED_EVT_IDX
REPORTED_ERT	Est rest time reported to caller.	INC. REPORTED_EST_REST_TIME	REPORTED_ERT
SHORT_DESC	Short description of trouble.	INC.SHORT_DESC	SHORT_DESC
TROUBLE_LOC	Incident's trouble location.	INC.TROUBLE_LOC	TROUBLE_LOC
UPDATE_EXISTING_INC	Whether to update an existing incident. If set to 1, then JMService will replace an existing incident for the same customer with the values passed in this call.		UPDATE_EXISTING_INC
USER_NAME	Call-taker user name. The name of the call-taker or interface that created the call.	INC.USER_NAME	USER_NAME
X_REF	Customer X coordinate. X coordinate of customer or customer device.	INC.X_COORD	X_REF
Y_REF	Customer Y coordinate. Y coordinate of customer or customer device.	INC.Y_COORD	Y_REF

During initialization of IVRAdapter, TROUBLE_CALLS column are matched with the standard product column (JMS.h). If TROUBLE_CALLS field does not match, error will be logged and IVRAdapter will exit.

The following are some exceptions when matching TROUBLE_CALLS columns with JMS Input String:

- TROUBLE_CALLS.CALL_COMMENT – JMS Input String COMMENT
- TROUBLE_CALLS.CALL_STATUS – special column in TROUBLE_CALLS table that indicates that the call is new (N) or already processed (C).
- TROUBLE_CALLS.SUPPLY_ID – if this column exist it replaces the value of TROUBLE_CALLS. CUST_DEVICE_IDX and TROUBLE_CALLS. CUST_DEVICE_CLS is set to 994.

Mapping to Customer-Defined Fields in Oracle Utilities Network Management System's INCIDENTS table

A configurable TROUBLE_CALLS column can also be done through SRS_RULES.

The following are the steps to map a new field in the TROUBLE_CALLS table with a new field in the INCIDENTS table:

1. Change the TROUBLE_CALLS table schema to include the customized field, for instance, TC_FIELD_ONE.
2. Change the INCIDENTS table schema to include a new field that will be mapped to TC_FIELD_ONE. For instance the new field on the INCIDENTS table would be INC_FIELD_ONE.
3. Create a new SRS Rule that maps the '201' (TROUBLE_CALLS reserve name) with the new field in the INCIDENTS table, INC_FIELD_ONE. See the **Map Customer-Defined Fields in the INCIDENTS table** on page 1-26 SRS rule for more information.
4. Restart JMService and the Generic IVR Adapter.

    Note: Before considering the option of introducing new fields in the TROUBLE_CALLS table and the INCIDENTS table, it is advisable to discuss such option with your Project Engineer.

Trouble Callback Mapping Configuration

IVR Adapter allows arbitrary information from the PICKLIST_INFO_UPD_TR table to be included into callback request. Columns CB_DETAIL1, CB_DETAIL2, CB_DETAIL3 and CB_DETAIL4 in the TROUBLE_CALLBACKS database table are used for this purpose. Database table IVR_ADAPTER_CONFIG is used to define if/how these columns should be populated.

Field Name	Nullable	Data Type	Description
CONFIG_ITEM	N	VARCHAR2(32)	Column name in the TROUBLE_CALLBACKS database table, which should be populated from the PICKLIST_INFO_UPD_TR table. Valid values are
CONFIG_VALUE	N	VARCHAR2(32)	Column name in the PICKLIST_INFO_UPD_TR table, which should be used as the data source.

SRS Rules Configuration

The following are SRS rules that could be used with the Generic IVR Adapter. These SRS rules can be included in the <project>_srs_rules.sql file.

Map Customer-Defined Fields in the INCIDENTS table

Oracle Utilities Network Management System and the Generic IVR Adapter provides a mechanism to receive additional trouble call information from the external application and have this information stored in a new customer-defined field in the INCIDENTS table of Oracle Utilities Network Management System.

The configurable TROUBLE_CALLS column name has special names '201', '202', up to '209' to serve this purpose. Normally, a regular column name in TROUBLE_CALLS like CALL_COMMENT or COMMENT in JMS Input String is tied to a specific field of the INCIDENTS table by default. For this case, it's the OP_COMMENT field. A special name like '201' could be mapped to a new field in the INCIDENTS table by using an SRS rule. The table below details how an SRS rule could be used to do this mapping. An SRS rule like this has to be used for each mapping. See **Map Customer-Defined Fields in the INCIDENTS table** on page 1-26 for more information.

Field Name	Value
SET_NAME	'config_incident'
INCIDENT_TYPE	'customer_defined'
RULE_NAME	the name of the new column in the INCIDENTS table
RULE_VALUE_1	'str', 'date', 'int', 'float' 'str' is for strings, 'date' is for dates, 'int' is for integers, and 'float' is for floats. This represents the data type of the new column.
RULE_VALUE_2	An integer, representing the value of the SRS input configuration item specifying this field -- must have a value of 200 or greater
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)
RULE_VALUE_INTEGER_5	0 (not used)

callbackInterfaceEnabled SRS Rule

If set to ‘yes’ then SRS APIs for manipulating callback information will become available. It has to be set to ‘yes’ for Web Callback GUI to operate. This rule holds outage information in JMService memory on a special data structure until this time expires OR all customer callbacks for the outage are complete.

Field Name	Value
SET_NAME	‘config’
INCIDENT_TYPE	‘any’
RULE_VALUE_1	‘yes’ or ‘no’ (default: ‘no’)
RULE_VALUE_2	0 (not used)
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)
RULE_VALUE_INTEGER_5	0 (not used)

useExternalCause SRS Rule

If set to ‘yes’ then the IVR Adapter’s callback requests data flow will include the cause code when it populates the TROUBLE_CALLBACKS table. The cause code value will be taken from JOBS.CAUSE.

Field Name	Value
SET_NAME	‘config’
INCIDENT_TYPE	‘any’
RULE_VALUE_1	‘yes’ or ‘no’ (default: ‘yes’)
RULE_VALUE_2	0 (not used)
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)
RULE_VALUE_INTEGER_5	0 (not used)

customerPhoneParentheses SRS Rule

If rule_value_1 set to 'yes', parentheses will be added to customer call phone numbers in the following format: **(AREA)NUMBER**. Otherwise, the number and area will be concatenated together without parentheses.

Field Name	Value
SET_NAME	'config'
INCIDENT_TYPE	'any'
RULE_VALUE_1	'yes' or 'no' (Default: 'yes')
RULE_VALUE_2	0 (not used)
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)
RULE_VALUE_INTEGER_5	0 (not used)

defaultCallbackAgent SRS Rule

Specifies username of the default callback agent. All new callbacks will be automatically assigned to this agent. If this rule is not set then new callbacks will be left unassigned.

Note: This rule only takes effect if the rule callbackInterfaceEnabled set to 'yes'. See **callbackInterfaceEnabled SRS Rule** on page 1-27 for more information. The agent name used should be considered as an external agent in the CES_USER and ENV_ACCESS tables. Also, it is recommended that 'IVR' be used as a value of RULE_VALUE_1, as this is the default callback agent name that the Generic IVR Adapter uses when the adapter runs.

Field Name	Value
SET_NAME	'config'
INCIDENT_TYPE	'any'
RULE_VALUE_1	callback agent username
RULE_VALUE_2	0 (not used)
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)
RULE_VALUE_INTEGER_5	0 (not used)

callbackFeederTimeout SRS Rule

The maximum time allowed (in minutes) between the current time and the restoration time of a resolution in callback module before the resolution is deemed too old to remain or be loaded into the callback module. This rule holds outage info in JMService memory in a special data structure until this time expires OR all customer callbacks for the outage are complete.

If this rule is set to 0 then resolutions will be kept in JMService until all callbacks are completed.

> **Note:** This rule **must** be used in conjunction with the **callbackInterfaceEnabled SRS Rule** on page 1-27.

Field Name	Value
INCIDENT_TYPE	'flowControlGeneral'
RULE_VALUE_1	"" (not used)
RULE_VALUE_2	An integer, representing a number of minutes (Default: 2880 -- *48 hours*)
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)
RULE_VALUE_INTEGER_5	0 (not used)

streetXsectionOffset SRS Rule

Specifies the size of the maximum bounding rectangle to be used in grouping street intersection fuzzy calls to supply nodes. The rectangle will be the area where:

```
x E [xsection_x - THIS RULE VALUE, xsection_x + THIS RULE VALUE]
                               and
y E [xsection_y - THIS RULE VALUE, xsection_y + THIS RULE VALUE]
```

This is an integer. Check your world coordinate system for a reasonable integer value.

This rule is used once for each rectangle desired (i.e., multiple instances of this rule may exist in a single rule set). For example, a larger rectangle size may be desired in a rural control zone and a smaller rectangle in an urban control zone.

Field Name	Value
RULE_VALUE_1	" (not used)
RULE_VALUE_2	integer (Default: none)
RULE_VALUE_INTEGER_1	0 (not used)
RULE_VALUE_INTEGER_2	0 (not used)
RULE_VALUE_INTEGER_3	0 (not used)
RULE_VALUE_INTEGER_4	0 (not used)

Field Name	Value
RULE_VALUE_INTEGER_5	0 (not used)
NCG_CLS	integer, representing ncg_cls of desired applicable control zone level
NCG_IDX	integer, representing ncg_idx of desired applicable control zone level

Generic IVR Adapter Trouble Call Performance

The maximum rate at which the Generic IVR Adapter injects trouble calls into the Oracle Utilities Network Management System is initially determined using the –callperpoll and –pollperiod command line parameters in the system.dat file. If these parameters are not set, the Generic IVR Adapter will, by default, retrieve a maximum of 100 trouble calls from the TROUBLE_CALLS table every six seconds and send these calls into the MMM via JMService. This corresponds to a maximum hourly call rate of 60,000 calls per hour.

If it is necessary to change this call rate while the adapter is running, two scripts are provided: ivrCallPerPoll.ces and ivrPollPeriod.ces. These scripts may be used to adjust the number of calls retrieved during each poll cycle and the period between poll cycles while the adapter is running.

> **Note**: If the adapter is restarted, these parameters (and the corresponding call rate) will revert to the command line parameters specified in the system.dat file (or the default values if no command line options are specified).

Command	Usage	Description
ivrCallPerPoll.ces	ivrCallPerPoll.ces NUM_CALLS_PER_POLL	Changes the number of calls retrieved from the TROUBLE_CALLS table during one poll cycle.
IvrPollPeriod.ces	IvrPollPeriod.ces NUM_SECONDS	Changes the period between poll cycles where calls are retrieved from the TROUBLE_CALLS table and submitted to JMService.

Generic IVR Adapter Troubleshooting

This section identifies high-level messages that could be sent to the Generic IVR Adapter using the Action command for troubleshooting purposes.

Command	Usage	Description
report	Action –services any.IVRGateway report	Reports back if the Generic IVR Adapter has started.
stop	Action –services any.IVRGateway stop	Stops the Generic IVR Adapter

Command	Usage	Description
debug	Action –services any.IVRGateway debug LEVEL (where LEVEL is 0, 1 or 2)	Sets the Generic IVR Adapter's debug level
cleantable	Action –services any.IVRGateway cleantable	Toggles the 'cleantable' command line option. Instructs if the Generic IVR Adapter should remove some records from its tables or not.

Note: It is important that the Generic IVR Adapter is already included in the System Data file to run high-level messages properly. For more information, see **Configure Adapter to run as NMS System Service** on page 1-9.

Database Schema

Overview

The following section defines in detail the schema of each database tables used by the Generic IVR Adapter. These Generic IVR Adapter tables however are not captured by performance mart. Moreover, this section defines the parameters used by the Generic IVR Adapter's stored procedures.

Database Table Schema

TROUBLE_CALLS Table Schema

The TROUBLE_CALLS table stores the trouble calls that are submitted by the external application. The Generic IVR Adapter polls this table and submits new trouble call records to Oracle Utilities Network Management System, so Oracle Utilities Network Management System could apply the outage analysis algorithm to predict the outage device. The external application indirectly inserts records to the TROUBLE_CALLS table by invoking the pr_trouble_calls stored procedure. See **pr_trouble_calls Stored Procedure** on page 1-47 for more information.

Each field of the TROUBLE_CALLS table is matched with SRSinput field. The mapping is configurable. A column names are directly tied up to a specific field of the INCIDENTS table or the JOBS table of Oracle Utilities Network Management System.

In effect, each field in the TROUBLE_CALLS table is mapped (and the mapping is configurable) to a particular field of the INCIDENTS table or the JOBS table of Oracle Utilities Network Management System. For more information, see **Trouble Call Mapping Configuration** on page 1-18.

In the 'Description' column, take note that field names prefixed with 'INC.' would come from the INCIDENTS table. Field names prefixed by 'JOBS.' would come from the JOBS table. Field names prefixed by 'CC.' would come from the CES_CUSTOMERS table.

Field Name	Nullable	Data Type	Description (JMS Input String reference)
ADDR_BUILDING	Y	VARCHAR2(10)	Customer building address. Refer to **ADDR_BUILDING** for more information. Map to INC.ADDR_BUILDING
ADDR_CITY	Y	VARCHAR2(45)	Customer City/State. Refer to **ADDR_CITY** for more information. Maps to INC.ADDR_CITY
ADDR_CROSS_STREET	Y	VARCHAR2(255)	Intersection cross street name. Maps to INC.ADDR_CROSS_STREET.
ADDR_STREET	Y	VARCHAR2(255)	Customer address. Refer to **ADDR_STREET** for more information. Maps to INC.ADDRESS and JOBS.ADDR_STREET

Field Name	Nullable	Data Type	Description (JMS Input String reference)
ALTERNATE_PHONE	Y	VARCHAR2(32)	Alternative contact number. Refer to **ALTERNATE_PHONE** for more information. Maps to INC.ALTERNATE_PHONE
APPT_RANGE	Y	NUMBER	Appointment Range. Refer to **APPT_RANGE** for more information. Maps to INC.APPT_RANGE.
APPT_TIME	Y	DATE	Time of appointment. Refer to **APPT_TIME** for more information. Maps to INC.APPT_TIME.
APPT_TYPE	Y	VARCHAR2(16)	Type of appointment. Refer to **APPT_TYPE** for more information. Maps to INC.APPT_TYPE.
CALL_COMMENT	Y	VARCHAR2(255)	Customer Comment. Refer to **COMMENT** Property Name for more information. Maps to INC.OP_COMMENT.
CALL_ID	Y	VARCHAR2(16)	Not used.
CALL_STATUS	Y	VARCHAR2(1)	Status of the trouble call in the TROUBLE_CALLS table. The Generic IVR Adapter uses this internally to identify the status of this trouble call. The possible values are as follows: 'N' - New trouble call 'I' - The Generic IVR Adapter is in the process of submitting this trouble call to Oracle Utilities Network Management System 'C' - Trouble call submission to Oracle Utilities Network Management System is completed. The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_CALLS table for 'old' records. Records with CALL_STATUS field = 'C' will be purged.
CALL_TIME	N	DATE	Input time of call. Refer to **CALL_TIME** for more information. Maps to INC.INPUT_TIME The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_CALLS table for 'old' records. The TROUBLE_CALL record is 'aged' based on the system date/time and the CALL_TIME field. Any record older than a predefined number of days will be removed. See **keepdbinfo** for more information.

Field Name	Nullable	Data Type	Description (JMS Input String reference)
CALL_TYPE	Y	VARCHAR2(8)	Type of call. Refer to **CALL_TYPE** for more information. Maps to INC.TYPE
CALLBACK_LATE	Y	VARCHAR2(1)	Callback late indicator. Refer to **CALLBACK_LATE** for more information. The possible values are as follows: 'Y' - It is OK to call back even when it is already late. 'N' - It is not OK to call back when it is already late. If no value was supplied, this field will default to 'N'. This information is only passed from the external application to Oracle Utilities Network Management System (using the Trouble Calls Data Flow), and back to the external application (using the Callback Requests Data Flow). No other action is taken.
CALLBACK_REQUEST	Y	NUMBER	Callback request indicator. Refer to **CALLBACK_REQUEST** for more information. The possible values are as follows: '0' - callback not requested '1' - callback requested Maps to INC.CALLBACK_REQUEST
CALLBACK_TIME	Y	DATE	Callback Before Time. Refer to **CALLBACK_TIME** for more information. Maps to INC.CALLBACK_TIME
CANCEL_CALL	Y	VARCHAR2(8)	Not used.
CHECK_CUTOFF	Y	VARCHAR2(1)	Check cut-off customer indicator. Refer to **CHECK_CUTOFF** for more information. The possible values are as follows: 'Y' - check if the customer is disconnected 'N' - do not perform checking.
CID_ALIAS	Y	VARCHAR2(32)	Not used.
CLUE	Y	NUMBER	Indicates if call is clue if set to Y. Refer to **CLUE** on page 1-20 for more information. Maps to INC.CLUE
COMBINE_PRI	Y	NUMBER	Total priority of call. Refer to **COMBINE_PRI** for more information.

Field Name	Nullable	Data Type	Description (JMS Input String reference)
CUST_ADDRESS	Y	VARCHAR2(200)	Not used.
CUST_CALL_CANCEL	Y	VARCHAR2(1)	Call cancel indicator. Refer to **CUST_CALL_CANCEL** for more information. Maps to INC.CALL_CANCEL
CUST_CITY	Y	VARCHAR2(25)	Not used.
CUST_CRITICAL	Y	VARCHAR2(1)	Critical customer indicator. This is added to the critical C count of the outage. Refer to **CUST_CRITICAL** for more information. Maps to INC.CRITICAL_CUST
CUST_DEVICE_ALIAS	Y	VARCHAR2(32)	Customer Device Alias. Refer to **CUST_DEVICE_ALIAS** for more information. Maps to INC.OBJECT
CUST_DEVICE_CLS	Y	NUMBER	Corresponding CC.H_CLS field for the given CC.SERV_LOC_ID. This field does not have a corresponding input parameter in the pr_trouble_calls stored procedure. The stored procedure itself populates this field. Refer to **CUST_DEVICE_CLS** for more information. Maps to INC.H_CLS
CUST_DEVICE_IDX	Y	NUMBER	Corresponding CC.H_IDX field for the given CC.SERV_LOC_ID. This field does not have a corresponding input parameter in the pr_trouble_calls stored procedure. The stored procedure itself populates this field. Refer to **CUST_DEVICE_IDX** for more information. Maps to INC.H_IDX
CUST_DEVICE_NCG	Y	NUMBER	NCG of customer device. Refer to **CUST_DEVICE_NCG** for more information. Maps to INC.NCG
CUST_DEVICE_PARTITION	Y	NUMBER	Partition of customer device. Refer to **CUST_DEVICE_ PARTITION** for more information. Maps to INC.PARTITION
CUST_FIRST_NAME	Y	VARCHAR2(75)	Customer Name. Refer to **CUST_FIRST_NAME** for more information. Maps to INC.CUSTOMER_NAME and JOBS.CUSTOMER_NAME
CUST_ID	Y	VARCHAR2(64)	Unique customer record identifier. Maps to INC.CID.

Field Name	Nullable	Data Type	Description <u>(JMS Input String reference)</u>
CUST_INTERSECT_CLS	Y	NUMBER	Intersecting device class. Refer to **CUST_INTERSECT_ CLS** for more information.
CUST_INTERSECT_IDX	Y	NUMBER	Intersecting device index. Refer to **CUST_INTERSECT_ IDX** for more information.
CUST_INTERSECT_NCG	Y	NUMBER	Intersecting NCG. Refer to **CUST_INTERSECT_ NCG** for more information.
CUST_INTR_X	Y	NUMBER	Intersecting X coordinate. X coordinate used for intersection grouping.
CUST_INTR_Y	Y	NUMBER	Intersecting Y coordinate.Y coordinate used for intersection grouping.
CUST_KEY	Y	VARCHAR2(16)	Corresponding CC.ACCOUNT_NUMBER field for the given CC.SERV_LOC_ID. This field does not have a corresponding input parameter in the pr_trouble_calls stored procedure. The stored procedure itself populates this field. Refer to **CUST_KEY** for more information. Maps to INC.ACCOUNT_NUM
CUST_LAST_NAME	Y	VARCHAR2(75)	The last name of the customer. Refer to **CUST_LAST_NAME** for more information. Maps to INC.CUSTOMER_NAME and JOBS.CUSTOMER_NAME
CUST_LIFE_SUPPORT	Y	VARCHAR2(1)	Life support customer. Refer to **CUST_LIFE_SUPPORT** for more information. Maps to INC.LIFE_SUPPORT
CUST_ORDER_NUM	Y	VARCHAR2(16)	Customer order number. Refer to **CUST_ORDER_NUM** for more information. Maps to INC.ORDER_NUMBER
CUST_PHONE	Y	VARCHAR2(32)	Customer phone number. Refer to **CUST_PHONE** for more information. Maps to INC.CUSTOMER_PHONE and JOBS.CUSTOMER_PHONE
CUST_PHONE_AREA	Y	VARCHAR2(8)	Customer phone area code. Refer to **CUST_PHONE_AREA** for more information. Maps to INC.CUSTOMER_PHONE and JOBS.CUSTOMER_PHONE

Field Name	Nullable	Data Type	Description (JMS Input String reference)
CUST_PHONE_UPDATE	Y	VARCHAR2(1)	Whether to update customer phone. Refer to **CUST_PHONE_ UPDATE** for more information.
CUST_PRIORITY	Y	VARCHAR2(4)	Customer Priority. Refer to **CUST_PRIORITY** for more information. This is defined by customer and needs to be an integer string. Maps to INC.CUSTOMER_TYPE
CUST_STATUS	Y	NUMBER	Condition status of call.
CUST_TROUBLE_CODE	N	VARCHAR2(10)	Trouble code or customer complaint. Refer to **CUST_TROUBLE_ CODE** for more information. This is the trouble or complaint that the customer reports when making a call. The trouble code determines the priority of the incident. Trouble code mapping setup in Oracle Utilities Network Management System should be synchronized with the trouble code mapping setup on the external application. This is to ensure that the trouble code sent from the external application is interpreted similarly when the trouble code is received by Oracle Utilities Network Management System. Maps to INC.COMPLAINT
CUST_TROUBLE_QUEUE	Y	VARCHAR2(10)	Customer trouble queue. Refer to **CUST_TROUBLE_ QUEUE** for more information. This field contains the name of the work group queue that the event has been referred to. Maps to INC.TROUBLE_QUEUE and JOBS.TROUBLE_QUEUE
DRV_INST	Y	VARCHAR2(180)	Driving instructions. Maps to INC.DRV_INSTR1
EXTERNAL_ID	N	VARCHAR2(16)	External ID. Refer to **EXTERNAL_ID** for more information If it is used, its value should be unique. Maps to INC.EXTERNAL_ID and JOBS.EXTERNAL_ID
FUZZY_NCG_CLS	Y	NUMBER	Fuzzy control zone class.
FUZZY_NCG_IDX	Y	NUMBER	Fuzzy control zone index.

Field Name	Nullable	Data Type	Description (JMS Input String reference)
GENERAL_AREA	Y	VARCHAR2(32)	General Area. Not Used in the SPL OMS System. Maps to INC.GENERAL_AREA
GROUP_BY_NAME	Y	VARCHAR2(127)	Fuzzy control zone name.
GROUPABLE	Y	NUMBER	Indicates if call is groupable if set to 1. Maps to INC.GROUPABLE
MEET_TIME	Y	DATE	Time of customer meet. Refer to **MEET_TIME** for more information. Maps to INC.MEET_TIME
MEET_TYPE	Y	NUMBER	Customer meet type. Refer to **MEET_TYPE** for more information. Maps to INC.MEET_CODE
METER_ID	Y	VARCHAR2(32)	Customer meter number. Maps to INC.METER_ID
POWER_UP	Y	NUMBER	Power-up call. Refer to POWER_UP for more information.
RELATED_EVT_APP	Y	NUMBER	Related event application.
RELATED_EVT_CLS	Y	NUMBER	Related event class. Maps to INC. RELATED_CLS
RELATED_EVT_IDX	Y	NUMBER	Related event index. Maps to INC. RELATED_IDX
REPORTED_ERT	Y	DATE	Estimated restoration time reported to caller. Maps to INC. REPORTED_EST_REST_TIME
SHORT_DESC	Y	VARCHAR2(128)	Trouble short description. Maps to INC.SHORT_DESC
TROUBLE_LOC	Y	VARCHAR2(255)	Incident's trouble location. Maps to INC.TROUBLE_LOC
UPDATE_EXISTING_INC	Y	NUMBER	Whether to update an existing incident. Refer to **UPDATE_EXISTING_ INC** for more information.
USER_NAME	Y	VARCHAR2(32)	Call-taker user name. Refer to **USER_NAME** for more information. Maps to INC.USER_NAME
X_REF	Y	NUMBER	Customer X coordinate. Refer to **X_REF** for more information. Maps to INC.X_COORD

Field Name	Nullable	Data Type	Description (JMS Input String reference)
Y_REF	Y	NUMBER	Customer Y coordinate. Refer to **Y_REF** for more information. Maps to INC.Y_COORD

TROUBLE_STATUS Table Schema

The Generic IVR Adapter inserts records to or updates records on the TROUBLE_STATUS table for each outage created, updated or merged within Oracle Utilities Network Management System. The external application can indirectly query this table to get outage status information using the pr_trouble_status stored procedure.

On the 'Description' column, take note that field names prefixed with 'JOBS.' would come from the JOBS table.

Field Name	Nullable	Data Type	Description
EVENT_CLS	N	NUMBER(38)	The value is taken from JOBS.EVENT_CLS
EVENT_IDX	N	NUMBER(38)	The value is taken from JOBS.EVENT_IDX
OUTAGE_STATUS	Y	VARCHAR2(32)	The value is taken from JOBS.ALARM_STATE. This is an abbreviation of the current state of the event, for instance, 'NEW', 'ASN', 'CMP', etc. Valid values are defined in the STATE_NAME field of the TE_VALID_STATES table.
OUTAGE_START_TIME	Y	DATE	The time of the lead call of the job. The value is taken from JOBS.FIRST_CALL_TIME
FIRST_DISPATCH_TIME	Y	DATE	The time the first crew was dispatched The value is taken from JOBS.FIRST_CREW_TIME
EST_RESTORE_TIME	Y	DATE	The last estimate of restoration time. The value is taken from JOBS.EST_REST_TIME

Field Name	Nullable	Data Type	Description
EST_RESTORE_TIME_SRC	Y	VARCHAR2(1)	The source of the ERT of the event. Possible values are as follows: 'N' - none (no ERT) 'S' - Oracle Utilities Network Management System – Storm Management 'P' - Oracle Utilities Network Management System Storm Management “non-published global ERT” 'O' - Oracle Utilities Network Management System Storm Management “onsite ERT” 'G' - Oracle Utilities Network Management System Storm Management “published global ERT” 'D' - Oracle Utilities Network Management System Storm Management "published global ERT delay" 'C' - User-entered (assumed to have been provided by the crew) 'I' - Initial default ERT 'M' - Oracle Utilities Network Management System Storm Management ERT is further in the future than allowed.
CREW_ARRIVAL_TIME	Y	DATE	The time when the crew arrived on location The value is taken from the WHEN_ARRIVED in the CREW_DISPATCHES table.
COMPLETION_TIME	Y	DATE	The value is taken from JOBS.JOB_COMPLETE_TIME. The time the event has been completed. This implies power restoration, the crew(s) are gone, and the event is completed in the Event Details window.
RESTORATION_TIME	Y	DATE	The value is taken from JOBS.RESTORE_TIME. The time that power has been restored. The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_STATUS table for 'old' records. The TROUBLE_STATUS record is 'aged' based on the system date/time and the RESTORATION_TIME field. Any record older than a predefined number of days will be removed. See **keepdbinfo** on page 1-14 for more information.

Field Name	Nullable	Data Type	Description
CASE_NOTE	Y	VARCHAR2(255)	The value is taken from JOBS.OPERATOR_COMMENT
STATUS	Y	NUMBER(38)	Condition status The value is taken from JOBS.STATUS. Valid values are defined in the TRANS_STATUS field of the TE_STATUSES table.
ACTIVE	Y	VARCHAR2(1)	The value is taken from JOBS.ACTIVE Possible values are as follows: 'Y' - Outage Is Active 'N' - Outage Is Not Active The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_STATUS table for 'old' records. Records with ACTIVE field = 'N' will be purged.
ALIAS	Y	VARCHAR2(64)	The device alias.
EVENT_TYPE	Y	VARCHAR2(32)	Possible values are as follows: OUT NON MEET PLAN SWP
FEEDER_NAME	Y	VARCHAR2(32)	The name of the feeder.
CAUSE	Y	VARCHAR2(32)	The cause of the outage if SRS Rule useExternalCause is on.
NUM_CALLS	Y	NUMBER	The number of calls.
NUM_CUST_OUT	Y	NUMBER	The number of customers out.

TROUBLE_AFFECTED_CUSTOMERS Table Schema

The Generic IVR Adapter updates the TROUBLE_AFFECTED_CUSTOMERS table for each customer affected by such outage. The Generic IVR Adapter provides the pr_affected_customers stored procedure that tells if a given customer is a part or has been a part of an outage, by querying the TROUBLE_STATUS and the TROUBLE_AFFECTED_CUSTOMERS tables.

he TROUBLE_AFFECTED _CUSTOMERS table is purged based on the purge mechanism used in the TROUBLE_STATUS table. Records in the TROUBLE_AFFECTED_CUSTOMER table with a corresponding purged event in the TROUBLE_STATUS table will be purged as well.

On the 'Description' column, take note that field names prefixed with 'SNL.' would come from the SUPPLY_NODES_LOG table. Field names prefixed by 'CC.' would come from the CES_CUSTOMERS table.

Field Name	Nullable	Data Type	Description
EVENT_CLS	N	NUMBER(38)	Class field of the event. Maps to SNL.EVENT_CLS
EVENT_IDX	N	NUMBER(38)	Index field of the event. Maps to SNL.EVENT_IDX
PREMISE_ID	N	VARCHAR2(50)	Maps to CC.SERV_LOC_ID
ACCOUNT_NUMBER	N	VARCHAR2(50)	Maps to CC.ACCOUNT_NUMBER
METER_ID	N	NUMBER(38)	Maps to CC.METER_ID
SUPPLY_NODE_IDX	N	NUMBER(38)	Handle index of the affected supply node. The value is taken from SNL.SUPPLY_IDX
CUSTOMER_NAME	Y	VARCHAR2(75)	Maps to CC.ACCOUNT_NAME
CUSTOMER_PHONE	Y	VARCHAR(38)	Maps to CC.PHONE_NUMBER
CUSTOMER_ADDRESS	Y	VARCHAR2(255)	Maps to CC.ADDRESS
RESTORE_TIME	Y	DATE	The value is taken from SNL.RESTORE_TIME

Since the CES_CUSTOMERS table varies on a project-to-project basis, consult your Project Engineer on what the expected values would be for the each field of the CES_CUSTOMERS table written above.

TROUBLE_CALLBACKS Table Schema

The TROUBLE_CALLBACKS table contains callback request information that has to be reported to the external application. The table also stores the corresponding callback response received from the external application. The Generic IVR Adapter directly inserts new callback requests to the said table. It also directly picks up processed callbacks from the same table. The external application is provided two stored procedures for indirectly reading and updating callback information from the table.

From the table below, on the 'Description' column, take note that field names prefixed with 'INC.' would come from the INCIDENTS table.

Column Name	Nullable	Data Type	Description
EVENT_CLS	Y	NUMBER(38)	Populated by the Callback Requests Data Flow from INC.EVENT_CLS.
EVENT_IDX	Y	NUMBER(38)	Populated by the Callback Requests Data Flow from INC.EVENT_IDX.
INCIDENT_NUMB	N	NUMBER(38)	Populated by the Callback Requests Data Flow from INC.NUMB.

Column Name	Nullable	Data Type	Description
PREMISE_ID	N	VARCHAR2(50)	Populated by the Callback Requests Data Flow from INC.ACCOUNT_NUM.
CUSTOMER_NAME	Y	VARCHAR2(75)	Populated by the Callback Requests Data Flow from INC.CUSTOMER_NAME.
CUSTOMER_PHONE	Y	VARCHAR2(38)	Populated by the Callback Requests Data Flow from INC.CUSTOMER_PHONE.
CUSTOMER_ADDRESS	Y	VARCHAR2(255)	Populated by the Callback Requests Data Flow by concatenating INC.ADDR_BUILDING, INC.ADDRESS and INC.ADDR_CITY
ALTERNATE_PHONE	Y	VARCHAR2(38)	Populated by the Callback Requests Data Flow from INC.ALTERNATE_PHONE.
TROUBLE_CODE	Y	VARCHAR2(32)	Populated by the Callback Requests Data Flow from INC.COMPLAINT. This is the trouble code (e.g., '10000000') of the incident rather than the clue (e.g., 'Out'). 'Out' is short for 'All Power Out'.
SHORT_DESCRIPTION	Y	VARCHAR2(128)	Populated by the Callback Requests Data Flow from INC.SHORT_DESC This is the clue (e.g., 'Out') of the incident rather than the trouble code (e.g., '10000000'). 'Out' is short for 'All Power Out'.
CUSTOMER_COMMENT	Y	VARCHAR2(255)	Populated by the Callback Requests Data Flow from INC.OP_COMMENT.
INCIDENT_TIME	Y	DATE	Populated by the Callback Requests Data Flow from INC.INPUT_TIME. The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_CALLBACKS table for 'old' records. The TROUBLE_CALLBACKS table record is 'aged' based on the system date/time and the INCIDENT_TIME field. Any record older than a predefined number of days will be removed. See **keepdbinfo** on page 1-14 for more information.
EXTERNAL_ID	Y	VARCHAR2(16)	Populated by the Callback Requests Data Flow from INC.EXTERNAL_ID.

Column Name	Nullable	Data Type	Description
CALLBACK_STATUS	Y	VARCHAR2(10)	Initially populated by the Callback Requests Data Flow as NULL; The field is repopulated by the external application (using pr_trouble_callback_responses stored procedure). The valid values are as follows: 'F' - Not Restored Callback 'R' - Restored Callback 'N' - Cancel Callback, unable to get a response The Callback Response Data Flow is responsible for sending the updated value to Oracle Utilities Network Management System. A remapped value is placed in INC.CALLBACK_STATUS.
CALLBACK_TIME	Y	DATE	Initially populated by the Callback Requests Data Flow as NULL; The field could be repopulated by the external application (using pr_trouble_callback_responses stored procedure). The stored procedure defaults this field to the system date if no information was supplied by the external application. The Callback Response Data Flow is responsible for sending the updated value to Oracle Utilities Network Management System. The value is placed in INC.CB_CALL_TIME.
CALL_TAKER_ID	Y	VARCHAR2(32)	Populated by the Callback Requests Data Flow from INC.USER_NAME.
CALLBACK_LATE	Y	VARCHAR2(1)	Populated by the Callback Requests Data Flow from INC.CALLBACK_LATE The possible values are as follows: ‘Y’ - It is OK to call back even when it is already late. ‘N’ - It is not OK to call back when it is already late. This information is only passed from the external application to Oracle Utilities Network Management System (using the Trouble Calls Data Flow), and back to the external application (using the Callback Requests Data Flow). No other action is taken.

Column Name	Nullable	Data Type	Description
CALLBACK_LATE_TIME	Y	DATE	Populated by the Callback Requests Data Flow from INC.CALLBACK_TIME. This information is only passed from the external application to Oracle Utilities Network Management System (using the Trouble Calls Data Flow), and back to the external application (using the Callback Requests Data Flow). No other action is taken.
CALLBACK_REASON	Y	VARCHAR2(100)	This is used by the Generic IVR Adapter to indicate the source of the callback request. This will default to 'OMS'.
PROCESS_STATUS	Y	VARCHAR2(1)	Initially populated by the Callback Requests Data Flow as 'N', signifying that the record is a new callback Once the record was fetched by the external application (using pr_trouble_callback_requests stored procedure), the field is automatically updated by the stored procedure to 'I' signifying that the external system is currently processing the callback response. As soon as the external application successfully returns the callback response to the Generic IVR Adapter (using pr_trouble_callback_responses stored procedure), the field is updated to 'C', signifying that the external application has completed the processing of the callback response. This field is internally maintained by the Generic IVR Adapter. Below is a list of valid values for this field. 'N' - New Callback 'I' - In Processing Of Callback Response 'C' - Completed The Processing Of Callback Response The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_CALLBACKS table for 'old' records. Records with PROCESS_STATUS field = 'C' will be purged.

Column Name	Nullable	Data Type	Description
CALLBACK_DONE	Y	VARCHAR2(1)	Initially populated by the Callback Requests Data Flow as 'N', signifying that the callback is not yet done. As soon as the external application successfully returns the callback response to the Generic IVR Adapter (using pr_trouble_callback_responses stored procedure), the field is updated to 'Y', signifying that the callback has been done. Below is a list of valid values for this field. 'N' - Callback Has Not Been Done 'Y' - Callback Has Been Done The Generic IVR Adapter uses this field as one of the criteria in purging the TROUBLE_CALLBACKS table for 'old' records. Records with CALLBACK_DONE field = 'Y' will be purged.
CAUSE_CODE	Y	VARCHAR2(32)	This is used to relay back to customers the cause of an outage when a callback is performed. Populated by the Callback Requests Data Flow from JOBS.CAUSE when the useExternalCause rule is set to ‘yes’ in the SRS_RULES.
OUTAGE_DURATION	Y	NUMBER	Outage duration in seconds. Populated by the Callback Requests Data with the difference between JOBS.RESTORE_TIME and JOBS.BEGIN_TIME.
CUSTOMER_COUNT	Y	NUMBER	Populated by the Callback Requests Data Flow from INC.USER_NAME.
CB_DETAIL_1	Y	VARCHAR2(80)	Populated by the Callback Requests Data Flow from a column in the PICKLIST_INFO_UPD_TR database table. Column name is configured in the IVR_ADAPTER_CONFIG database table.
CB_DETAIL_2	Y	VARCHAR2(80)	Populated by the Callback Requests Data Flow from a column in the PICKLIST_INFO_UPD_TR database table. Column name is configured in the IVR_ADAPTER_CONFIG database table.
CB_DETAIL_3	Y	VARCHAR2(80)	Populated by the Callback Requests Data Flow from a column in the PICKLIST_INFO_UPD_TR database table. Column name is configured in the IVR_ADAPTER_CONFIG database table.

Column Name	Nullable	Data Type	Description
CB_DETAIL_4	Y	VARCHAR2(80)	Populated by the Callback Requests Data Flow from a column in the PICKLIST_INFO_UPD_TR database table. Column name is configured in the IVR_ADAPTER_CONFIG database table.

Stored Procedure Parameters

pr_trouble_calls Stored Procedure

The Generic IVR Adapter provides the pr_trouble_calls procedure to be used by the external application to insert trouble calls in the TROUBLE_CALLS table. Refer to **Trouble Calls** on page 1-4 for Data Flow details.

Below is a high level description of what is done inside the stored procedure

- Upon invoking the stored procedure, the p_premise_id parameter is used to query the CES_CUSTOMERS table (via the SERV_LOC_ID field) to retrieve the ACCOUNT_NUMBER, H_CLS and H_IDX fields of the said table. The value of these fields is placed in the corresponding columns of the TROUBLE_CALLS table.
- Other parameter values are inserted to corresponding fields on the TROUBLE_CALL table.
- Several TROUBLE_CALLS columns will have default value when no parameter value is supplied.
- Should there be an error in the record insert, an Oracle error is returned.

  Note: If the given premise id has multiple accounts associated with it, only one account (i.e., the first account) is used.

Below are details about each parameter of the pr_trouble_calls stored procedure. Note that the field name column indicates the corresponding column that is populated in the TROUBLE_CALLS table.

Parameters

Parameter	Direction	Data Type	Field Name	Comment
p_premise_id	In	VARCHAR2	PREMISE_ID	The value is inserted as is.
p_trouble_code	In	VARCHAR2	TROUBLE_CODE	Defaults to ‘1’ followed by a certain number of ‘0’. If no value was supplied. The total length of the string is the total number of distinct groups in the SRS_TROUBLE_CODES table.

Parameter	Direction	Data Type	Field Name	Comment
p_callback_ind	In	VARCHAR2	CALLBACK_INDICATOR	The possible values are as follows: '0' - callback not requested '1' - callback requested Defaults to '1' if no value is supplied. 'Y' is translated to '1'. 'N' is translated to '0'.
p_call_time	In	DATE	CALL_TIME	Defaults to the database system date if no value is supplied
p_call_taker_id	In	VARCHAR2	CALL_TAKER_ID	The value is inserted as is.
p_alternate_phone	In	VARCHAR2	ALTERNATE_PHONE	The value is inserted as is.
p_customer_comment	In	VARCHAR2	CUSTOMER_COMMENT	The value is inserted as is.
p_customer_phone	In	VARCHAR2	CUSTOMER_PHONE	The value is inserted as is.
p_customer_name	In	VARCHAR2	CUSTOMER_NAME	The value is inserted as is.
p_customer_address	In	VARCHAR2	CUSTOMER_ADDRESS	The value is inserted as is.
p_customer_city_state	In	VARCHAR2	CUSTOMER_CITY_STATE	The value is inserted as is.
p_customer_priority	In	VARCHAR2	CUSTOMER_PRIORITY	The value is inserted as is.
p_external_id	In	VARCHAR2	EXTERNAL_ID	The value is inserted as is.
p_device_alias	In	VARCHAR2	DEVICE_ALIAS	The value is inserted as is.
p_check_cutoff_ind	In	VARCHAR2	CHECK_CUTOFF_IND	The possible values are as follows: 'Y' - check if the customer is disconnected 'N' - do not perform checking. Defaults to 'N' if no value is supplied
p_callback_late_ind	In	VARCHAR2	CALLBACK_LATE_IND	The possible values are as follows: 'Y' - It is OK to call back even when it is already late. 'N' - It is not OK to call back when it is already late. Defaults to 'N' if no value is supplied
p_callback_before_time	In	DATE	CALLBACK_BEFORE_TIME	The value is inserted as is.
p_trouble_queue	In	VARCHAR2	TROUBLE_QUEUE	The value is inserted as is.

Parameter	Direction	Data Type	Field Name	Comment
p_meter_id	In	VARCHAR2	METER_ID	The value is inserted as is.
p_supply_id	In	NUMBER	SUPPLY_ID	The value is inserted as is.
p_cust_phone_area	In	VARCHAR2	CUST_PHONE_AREA	The value is inserted as is.
p_cust_last_name	In	VARCHAR2	CUST_LAST_NAME	The value is inserted as is.
p_general_area	In	VARCHAR2	GENERAL_AREA	The value is inserted as is.
p_cust_order_num	In	VARCHAR2	CUST_ORDER_NUM	The value is inserted as is.
p_drv_inst	In	VARCHAR2	DRV_INST	The value is inserted as is.
p_cust_life_support	In	VARCHAR2	CUST_LIFE_SUPPORT	The value is inserted as is.
p_cust_call_cancel	In	VARCHAR2	CUST_CALL_CANCEL	The value is inserted as is.
p_short_desc	In	VARCHAR2	SHORT_DESC	The value is inserted as is.
p_addr_building	In	VARCHAR2	ADDR_BUILDING	The value is inserted as is.
p_meet_time	In	DATE	MEET_TIME	The value is inserted as is.
p_meet_type	In	NUMBER	MEET_TYPE	The value is inserted as is.
p_groupable	In	NUMBER	GROUPABLE	The value is inserted as is.
p_clue	In	NUMBER	CLUE	The value is inserted as is.
p_combine_pri	In	NUMBER	COMBINE_PRI	The value is inserted as is.
p_cust_status	In	NUMBER	CUST_STATUS	The value is inserted as is.
p_cust_intr_x	In	NUMBER	CUST_INTR_X	The value is inserted as is.
p_cust_intr_y	In	NUMBER	CUST_INTR_Y	The value is inserted as is.
p_cust_intersect_cls	In	NUMBER	CUST_INTERSECT_CLS	The value is inserted as is.
p_cust_intersect_idx	In	NUMBER	CUST_INTERSECT_IDX	The value is inserted as is.
p_cust_intersect_ncg	In	NUMBER	CUST_INTERSECT_NCG	The value is inserted as is.
p_update_existing_inc	In	NUMBER	UPDATE_EXISTING_INC	The value is inserted as is.
p_fuzzy_ncg_cls	In	NUMBER	FUZZY_NCG_CLS	The value is inserted as is.
p_fuzzy_ncg_idx	In	NUMBER	FUZZY_NCG_IDX	The value is inserted as is.
p_group_by_name	In	VARCHAR2	GROUP_BY_NAME	The value is inserted as is.
p_cust_critical	In	VARCHAR2	CUST_CRITICAL	The value is inserted as is.
p_related_evt_cls	In	NUMBER	RELATED_EVT_CLS	The value is inserted as is.
p_related_evt_idx	In	NUMBER	RELATED_EVT_IDX	The value is inserted as is.
p_related_evt_app	In	NUMBER	RELATED_EVT_APP	The value is inserted as is.
p_x_ref	In	NUMBER	X_REF	The value is inserted as is.

Parameter	Direction	Data Type	Field Name	Comment
p_y_ref	In	NUMBER	Y_REF	The value is inserted as is.
p_call_type	In	VARCHAR2	CALL_TYPE	The value is inserted as is.
p_cust_phone_update	In	VARCHAR2	CUST_PHONE_UPDATE	The value is inserted as is.
p_trouble_loc	In	VARCHAR2	TROUBLE_LOC	The value is inserted as is.
p_appt_type	In	VARCHAR2	APPT_TYPE	The value is inserted as is.
p_appt_time	In	DATE	APPT_TIME	The value is inserted as is.
p_appt_range	In	NUMBER	APPT_RANGE	The value is inserted as is.
p_cust_device_ncg	In	NUMBER	CUST_DEVICE_NCG	The value is inserted as is.
p_cust_device_partition	In	NUMBER	CUST_DEVICE_PARTITION	The value is inserted as is.
p_err_premise_id	Out	VARCHAR2	VARCHAR2(80)	The erroneous premise ID input parameter
p_err_oracle_error	Out	VARCHAR2	VARCHAR2(80)	Oracle's error message.

Note: The pr_trouble_calls stored procedure does not require a call status parameter from the user to insert in the TROUBLE_CALLS stored procedure. Each time the stored procedure inserts trouble calls in the TROUBLE_CALLS table, the CALL_STATUS field is always 'N', signifying that it is a new trouble call.

pr_trouble_status Stored Procedure

The Generic IVR Gateway provides the pr_trouble_status stored procedure that queries outage information for a given event handle parameter. When the -eventstatus command line option is used, the data is retrieved from the TROUBLE_STATUS table by calling the stored procedure pr_table_trouble_status. Refer to the **Event Status** on page 1-5 for Data Flow details. Otherwise, the stored procedure pr_direct_trouble_status is called, which retrieves data directly from the Oracle Utilities Network Management System tables.

Below is a high level description of what is done inside the stored procedure:

- If a single row was retrieved, the rest of the stored procedure parameters are populated with data retrieved from either the TROUBLE_STATUS table or Oracle Utilities Network Management System tables.
- Else, an Oracle error is returned.

Below are details about each parameter of the pr_trouble_status stored procedure. Note that the field name column indicates the corresponding column in the TROUBLE_STATUS table.

Parameters

Parameter	Direction	Data Type	Field Name	Comments
p_in_event_class	In	NUMBER	EVENT_CLS	Event class input parameter
p_in_event_index	In	NUMBER	EVENT_IDX	Event index input parameter
p_out_event_class	Out	NUMBER	EVENT_CLS	Similar to the event class input parameter
p_out_event_index	Out	NUMBER	EVENT_IDX	Similar to the event index input parameter
p_outage_status	Out	VARCHAR2	OUTAGE_STATUS	This is an abbreviation of the current state of the event, for instance, 'NEW', 'ASN', 'CMP', etc.
p_outage_start_time	Out	DATE	OUTAGE_START_TIME	The time of the lead call of the job.
p_first_dispatch_time	Out	DATE	FIRST_DISPATCH_TIME	The time the first crew was dispatched
p_est_restore_time	Out	DATE	EST_RESTORE_TIME	The last estimate of restoration time.

Parameter	Direction	Data Type	Field Name	Comments
p_est_restore_time_src	Out	VARCHAR2	EST_RESTORE_TIME_SRC	The source of the ERT of the event. Possible values are as follows: 'N' - none (no ERT) 'S' - Storm Management 'P' - Storm Management "non-published global ERT" 'O' - Storm Management "onsite ERT" 'G' - Storm Management "published global ERT" 'D' - Storm Management "publishedglobalERTdelay" 'C' - User-entered (assumed to have been provided by the crew) 'I' - Initial default ERT 'M' - Storm Management ERT is further in the future than allowed.
p_crew_arrival_time	Out	DATE	CREW_ARRIVAL_TIME	The time when the crew arrived on location
p_completion_time	Out	DATE	COMPLETION_TIME	The time the event has been completed. This implies power restoration, the crew(s) are gone, and the event is completed in the Event Details window.
p_restoration_time	Out	DATE	RESTORATION_TIME	The time that power has been restored.
p_case_note	Out	VARCHAR2	CASE_NOTE	Comment
p_status	Out	NUMBER	STATUS	Condition status
p_active	Out	VARCHAR2	ACTIVE	Possible values are as follows: 'Y' - Outage Is Active 'N' - Outage Is Not Active
p_alias	Out	VARCHAR2	ALIAS	The device alias.
p_event_type	Out	VARCHAR2	EVENT_TYPE	Possible values are as follows: OUT NON MEET PLAN SWP
p_feeder_name	Out	VARCHAR2	FEEDER_NAME	The name of the feeder.

Parameter	Direction	Data Type	Field Name	Comments
p_cause	Out	VARCHAR2	CAUSE	The cause of the outage if SRS Rule useExternalCause is on.
p_num_calls	Out	NUMBER	NUM_CALLS	The number of calls.
p_num_cust_out	Out	NUMBER	NUM_CUST_OUT	The number of customers out.
p_err_event_class	Out	NUMBER		The erroneous event class input parameter.
p_err_event_index	Out	NUMBER		The erroneous event index input parameter.
p_err_oracle_error	Out	VARCHAR2		Oracle's error message.

pr_affected_customers Stored Procedure

The Generic IVR Adapter provides the pr_affected_customers stored procedure that tells if a given customer has been a part of an outage. When the -eventstatus command line option is used, the data is retrieved from the TROUBLE_STATUS and TROUBLE_AFFECTED_CUSTOMERS tables by calling the stored procedure pr_table_affected_customers. Refer to the data flow for **Affected Customers** on page 1-5. Otherwise, the stored procedure pr_direct_affected_customers is called, which retrieves data directly from the Oracle Utilities Network Management System tables.

Below is a high level description of what occurs inside the stored procedure.

- The stored procedure tries to get the latest **active** event for the given premise ID (p_in_premise_id input parameter). The retrieved event handle is used as a parameter for the pr_trouble_status stored procedure. After which, the pr_affected_customers stored procedure returns data retrieved from the TROUBLE_AFFECTED_CUSTOMERS table and the pr_trouble_status stored procedure.
- If the customer is not part of an **active** event, the stored procedure tries to get the latest inactive event for the given premise ID. The retrieved event handle is used as a parameter for the pr_trouble_status stored procedure. After which, the pr_affected_customers stored procedure returns data retrieved from the TROUBLE_AFFECTED_CUSTOMERS table and the pr_trouble_status stored procedure.
- If the customer is not involved in any active or inactive events, all output parameters are returned as null for the parameter's corresponding data type.

Parameters

Below are details about each parameter of the pr_affected_customers stored procedure. Take note that the field name column would be field names prefixed with 'TS.' if the field name came from the TROUBLE_STATUS table. Field names would be prefixed with 'TAC.' if the field name came from the TROUBLE_AFFECTED_CUSTOMERS table. Field names would be prefixed with CC.' if the field name came from the CES_CUSTOMERS table.

Parameter	Direction	Data Type	Field Name	Comments
p_in_premise_id	In	VARCHAR2		Premise ID input parameter with a corresponding entry in CC.SERV_LOC_ID
p_out_premise_id	Out	VARCHAR2	TAC.PREMISE_ID	Similar to the Premise ID input parameter
p_account_number	Out	VARCHAR2	TAC.ACCOUNT_NUMBER	
p_customer_name	Out	VARCHAR2	TAC.CUSTOMER_NAME	
p_customer_phone	Out	VARCHAR2	TAC.CUSTOMER_PHONE	
p_customer_address	Out	VARCHAR2	TAC.CUSTOMER_ADDRESS	
p_supply_node_index	Out	NUMBER	TAC.SUPPLY_NODE_IDX	
p_supply_node_restore_time	Out	DATE	TAC.RESTORE_TIME	
p_event_class	Out	NUMBER	TS.EVENT_CLS	The event class of the most recent active (or most recent inactive) event retrieved using the premise ID.
p_event_index	Out	NUMBER	TS.EVENT_IDX	The event index of the most recent active (or most recent inactive) event retrieved using the premise ID.
p_outage_status	Out	VARCHAR2	TS.OUTAGE_STATUS	This is an abbreviation of the current state of the event, for instance, 'NEW', 'ASN', 'CMP', etc.
p_outage_start_time	Out	DATE	TS.OUTAGE_START_TIME	The time of the lead call of the job.
p_first_dispatch_time	Out	DATE	TS.FIRST_DISPATCH_TIME	The time the first crew was dispatched
p_est_restore_time	Out	DATE	TS.EST_RESTORE_TIME	The last estimate of restore time.

Parameter	Direction	Data Type	Field Name	Comments
p_est_restore_time_ src	Out	VARCHAR2	TS.EST_RESTORE_TIME_ SRC	The source of the ERT of the event. Possible values are as follows: 'N' - none (no ERT) 'S' - Storm Management 'P' - Storm Management “non-published global ERT” 'O' - Storm Management “onsite ERT” 'G' - Storm Management “published global ERT” 'D' - Storm Management "published global ERT delay" 'C' - User-entered (assumed to have been provided by the crew) 'I' - Initial default ERT 'M' - Storm Management ERT is further in the future then allowed
p_crew_arrival_time	Out	DATE	TS.CREW_ARRIVAL_ TIME	The time when the crew arrived on location
p_completion_time	Out	DATE	TS.COMPLETION_TIME	The time the event has been completed. This implies power restoration, the crew(s) are gone, and the event is completed in the Event Details window.
p_restoration_time	Out	DATE	TS.RESTORATION_TIME	The time that power has been restored.
p_case_note	Out	VARCHAR2	TS.CASE_NOTE	Comment
p_status	Out	NUMBER	TS.STATUS	Condition status
p_active	Out	VARCHAR2	TS.ACTIVE	Possible values are as follows: 'Y' - Outage Is Active 'N' - Outage Is Not Active

pr_trouble_callback_requests Stored Procedure

Below is a high level description of what is done inside the stored procedure

- From the TROUBLE_CALLBACKS table, a list of new callback requests is created. These are the TROUBLE_CALLBACKS records whose PROCESS_STATUS field is 'N' (New) and CALLBACK_DONE field is 'N' (No).
- The list is captured within the stored procedure as a database cursor and returned to the calling application.
- The PROCESS_STATUS field of the records in the list is updated from 'N' (New) to 'I' (In Progress).

Note: Refer to the Data Flow Steps of the Callback Requests Data Flow on how the TROUBLE_CALLBACKS table is populated.

Parameter

Parameter	Direction	Cursor
p_callback_requests	In/Out	CALLBACK_CURSOR

Cursor Definition

Below are the fields of the CALLBACK_CURSOR. Take note that the CALLBACK_CURSOR is defined as a weakly typed cursor.

Field Name from the Cursor	Data Type	Field Name from TROUBLE_CALLBACKS	Comments
EVENT_CLS	NUMBER(38)	TCB.EVENT_CLS	Event class
EVENT_IDX	NUMBER(38)	TCB.EVENT_IDX	Event index
INCIDENT_NUMB	NUMBER(38)	TCB.INCIDENT_NUMB	Incident number
PREMISE_ID	VARCHAR2(50)	TCB.PREMISE_ID	Premise id
CUSTOMER_NAME	VARCHAR2(75)	TCB.CUSTOMER_NAME	Customer name
CUSTOMER_PHONE	VARCHAR2(38)	TCB.CUSTOMER_PHONE	Customer phone
CUSTOMER_ADDRESS	VARCHAR2(255)	TCB.CUSTOMER_ADDRESS	Customer address
ALTERNATE_PHONE	VARCHAR2(38)	TCB.ALTERNATE_PHONE	Customer alternate phone number
TROUBLE_CODE	VARCHAR2(32)	TCB.TROUBLE_CODE	This is the trouble code (e.g., '10000000') of the incident rather than the clue (e.g., 'Out'). 'Out' is short for 'All Power Out'.
SHORT_DESCRIPTION	VARCHAR2(128)	TCB.SHORT_DESCRIPTION	This is the clue (e.g., 'Out') of the incident rather than the trouble code (e.g., '10000000'). 'Out' is short for 'All Power Out'.

Field Name from the Cursor	Data Type	Field Name from TROUBLE_CALLBACKS	Comments
CUSTOMER_COMMENT	VARCHAR2(255)	TCB.CUSTOMER_COMMENT	Call-taker Comments. Comments provided by the customer or call-taker about the incident.
INCIDENT_TIME	DATE	TCB.INCIDENT_TIME	Input time of call. The input time of the incident.
EXTERNAL_ID	VARCHAR2(16)	TCB.EXTERNAL_ID	Unique call identifier. The unique identifier for the incident.
CALL_TAKER_ID	VARCHAR2(32)	TCB.CALL_TAKER_ID	Call-taker user name. The name of the call-taker or interface that created the call.
CALLBACK_LATE	VARCHAR2(1)	TCB.CALLBACK_LATE	The possible values are as follows: 'Y' - It is OK to call back even when it is already late. 'N' - It is not OK to call back when it is already late.
CALLBACK_LATE_TIME	DATE	TCB.CALLBACK_LATE_TIME	
CALLBACK_REASON	VARCHAR2(100)	TCB.CALLBACK_REASON	This will default to 'OMS'.
CAUSE_CODE	VARCHAR2(32)	TCB.CAUSE_CODE	Cause code of the event related to the callback.

pr_trouble_callback_responses Stored Procedure

Below is a high level description of what is done inside the stored procedure

- Upon receiving the input parameter values, the stored procedure verifies if either the p_incident_numb input parameter or the p_external_id input parameter was supplied. If both were supplied, the p_incident_numb parameter takes precedence.
- The stored procedure validates if the p_callback_status input parameter has a valid value. The valid values are 'F' (not restored), 'R' (restored) and 'N' (cancel callback).
- The stored procedure verifies that there is a unique combination of p_incident_numb and p_premise_id OR a unique combination of p_external_id and p_premise_id on the TROUBLE_CALLBACKS table, whichever among p_incident_numb or p_external_id was supplied.
- The TROUBLE_CALLBACKS table is updated for the p_incident_numb and p_premise_id combination OR the p_external_id and p_premise_id combination. The following fields are updated:
    - The callback's CALLBACK_DONE field to 'Y' signifying that the callback was already done.
    - The callback's CALLBACK_TIME field with provided p_callback_time stored procedure parameter. CALLBACK_TIME field defaults to the system date if no value was provided.
    - The callback's CALLBACK_STATUS field with the appropriate callback response code.

- Should any of these steps fail, the stored procedure exits and returns the appropriate error.

  Note: Refer to the Data Flow Steps of the Callback Response Data Flow on how the TROUBLE_CALLBACKS table is populated.

Parameters

Parameter	Direction	Data Type	Field Name	Comments
p_incident_numb	In	NUMBER	INCIDENT_NUMB	Incident Number. Either this or the p_external_id parameter has to be supplied
p_external_id	In	VARCHAR2	EXTERNAL_ID	External Id. Either this or the p_incident_numb parameter has to be supplied
p_premise_id	In	VARCHAR2	PREMISE_ID	Premise Id.
p_callback_status	In	VARCHAR2	CALLBACK_STATUS	The valid values are as follows: 'F' - Not Restored Callback 'R' - Restored Callback 'N' - Cancel Callback, unable to get a response
p_callback_time	In	DATE	CALLBACK_TIME	Defaulted to the system date if no value was supplied
p_err_incident_numb	Out	NUMBER		The erroneous incident number input parameter
p_err_external_id	Out	VARCHAR2		The erroneous external ID input parameter
p_err_premise_id	Out	VARCHAR2		The erroneous premise ID input parameter
p_err_oracle_error	Out	VARCHAR2		Oracle's error message

pr_customer_event_details Stored Procedure

The Generic IVR Gateway provides the pr_customer_event_details stored procedure that gives the event details of an outage given the customer premise using the pr_trouble_status stored procedure. Refer to the data flow detail for **Affected Customers** on page 1-5.

Below is a high level description of what is done inside the stored procedure.

- The stored procedure tries to get the latest event for the given premise ID (p_in_premise_id input parameter). The retrieved event handle is used as a parameter for the pr_trouble_status stored procedure. After which, the pr_customer_event_details stored procedure returns the same information returned by pr_trouble_status stored procedure.

Parameters

Below are details about each parameter of the pr_customer_event_details stored procedure.

Parameter	Direction	Data Type	Comments
p_in_premise_id	In	VARCHAR2	Premise ID input parameter with a corresponding entry in CES_CUSTOMERS.SERV_LOC_ID
p_out_event_class	Out	NUMBER	Event class output parameter
p_out_event_index	Out	NUMBER	Event index output parameter
p_out_outage_status	Out	VARCHAR2	This is an abbreviation of the current state of the event, for instance, 'NEW', 'ASN', 'CMP', etc.
p_ out_outage_start_time	Out	DATE	The time of the lead call of the job.
p_ out_first_dispatch_time	Out	DATE	The time the first crew was dispatched
p_ out_est_restore_time	Out	DATE	The last estimate of restoration time.
p_ out_est_restore_time_src	Out	VARCHAR2	The source of the ERT of the event. Possible values are as follows: 'N' - none (no ERT) 'S' - Storm Management 'P' - Storm Management "non-published global ERT" 'O' - Storm Management "onsite ERT" 'G' - Storm Management "published global ERT" D' - Storm Management "published global ERT delay" 'C' - User-entered (assumed to have been provided by the crew) 'I' - Initial default ERT 'M' - Storm Management ERT is further in the future then allowed
p_ out_crew_arrival_time	Out	DATE	The time when the crew arrived on location
p_ out_completion_time	Out	DATE	The time the event has been completed. This implies power restoration, the crew(s) are gone, and the event is completed in the Event Details window.
p_ out_restoration_time	Out	DATE	The time that power has been restored.
p_ out_case_note	Out	VARCHAR2	Comment
p_ out_status	Out	NUMBER	Condition status

Parameter	Direction	Data Type	Comments
p_ out_active	Out	VARCHAR2	Possible values are as follows: 'Y' - Outage Is Active 'N' - Outage Is Not Active
p_out_alias	Out	VARCHAR2	The device alias.
P_out_event_type	Out	VARCHAR2	Possible values are as follows: OUT NON MEET PLAN SWP
P_out_feeder_name	Out	VARCHAR2	The name of the feeder.
P_out_cause	Out	VARCHAR2	The cause of the outage if the SRS Rule useExternalCause is on.
P_out_num_calls	Out	NUMBER	The number of calls.
P_out_num_cust_out	Out	NUMBER	The number of customers out.
P_err_premise_id	Out	VARCHAR2	
P_err_oracle_error	Out	VARCHAR2	

Terminology

The following terms and acronyms are relevant to this specification

OMS	Outage Management System
Network Management System	Network Management System
CIS	Customer Information System
IVR	Interactive Voice Response
Generic IVR Adapter	A Unix application that generally executes on the OMS server machine. It supports the Trouble Call, Event Status, Affected Customers, Callback Request and Callback Response Data Flows.
SMService	System Monitor Service. SMService monitors the core processes in the system, essentially the services and interfaces.
JMService	Job Management Service. The Oracle Utilities Network Management System call processing and outage prediction engine.
ODService	Object Directory Service. ODService improves performance of the Oracle Utilities Network Management System by caching large amounts of device information that is likely to be requested by applications. This caching allows the requests to be handled very quickly without directly accessing the database.
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 and comprises the backbone of the system, providing access to the server for each client and the communication required between tools and services. Isis delivers the organized information to the client applications.

Chapter 2

Generic SCADA Adapter

This chapter includes the following topics:

- **Introduction**
- **Generic SCADA Adapter Configuration**
- **Software Configuration**
- **Data Flows**
- **Information Model**

Introduction

The Generic SCADA Adapter (executable name: RTAdapter) is an Oracle Utilities Network Management System interfacing adapter that can be used to interface to external SCADA systems. Generally one RTAdapter process is configured to communicate with each external SCADA system. Each external SCADA system must write some form of "scan file" to a defined (dedicated) directory (or RDBMS table) that RTAdapter can access in read/write mode. The "scan files" are read out of the named directory (or RDBMS table) by RTAdapter on a first-in-first-out basis (to support queuing of updates). The form of the scan files can be configured (to some degree) for each SCADA interface.

Generic SCADA Adapter Configuration

Overview

This section is used to guide the user in the configuration of the Oracle Utilities Network Management System Generic SCADA Adapter. The following are assumed to be true before the adapter is installed:

- Oracle database access has been confirmed.
- ISIS messaging bus has been installed and verified.
- Oracle Utilities Network Management System is installed and functional.

Configure Adapter to Run as an NMS System Service

Configure the Generic SCADA Adapter to run as an Oracle Utilities Network Management System service by updating the `$NMS_HOME/etc/system.dat` file to include the Generic SCADA Adapter as a system service. There are three main sections where this service needs to be defined: the service, program, and instance sections.

See the `$CES_HOME/templates/system.dat.template` file for examples of how to configure the Generic SCADA Adapter. Search for USA (Universal SCADA Adapter) in the file and copy those lines to `$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 Generic SCADA Adapter to properly be monitored by SMService.

Notes:

- The adapter process is generally given a configuration name adapted from the SCADA system from which it is receiving data (such as the USA name used in the system.dat.template file noted above). Also, take note that in the example above, it is assumed that the Generic SCADA Adapter will reside on the same machine where the Oracle Utilities Network Management System environment resides.
- If the generic SCADA adapter is configured to use the "-dir RDBMS" option (where RTAdapter periodically scans the scada_digital_in and scada_analog_in tables for updates), it is recommended that a dedicated DBService instance by configured to support the RTAdapter. To do this you must configure RTAdapter with the -rtdbs option and you MUST configure an additional DBService process in your system.dat with the following program entry. You will also need appropriate corresponding "service" and "instance" records in your system.dat file.

```
program  RTDBService     DBService       -nodaemon
    -service rt   -process_name RTDBService
```

Command Line Options for Generic SCADA Adapter

The command line for the Generic SCADA Adapter provides the following options:

Command Line Option	What it does
-nodaemon	Run in foreground - when running by hand.
-volbuffer <VOLTS>	Threshold for analog VOLT attributes, report if changed by more than <VOLTS>.
-ampbuffer <AMPS>	Threshold for analog AMP attributes, report if changed by more than <AMPS>.
-idle <cycles>	Number of processing cycles to wait without processing any data before sending an alarm.
-lock	Use file locking to prevent file overwrite during file read (if scan file names are not unique).
-error	Show processing errors.
-watch	Show files being processed, minimal info.
-debug	Enable debugging.
-operate	Operate (change status of) model devices, otherwise generates pseudo alarms only.
-controls	Call the project/SCADA specific rti_project.ces script when an outbound control request (open/close) is made. This script can be customized to help trigger an action (some type of control request) on external system. Generally for testing.
-offline	For testing purposes - when a control request is received - simulate the external SCADA system and operate the device.
-dir <directory>	Value either directing RTAdapter to the directory containing scada data scan files or if set to 'RDBMS' enabling the scada_digital_in/scada_analog_in database polling functionality. (Required)
-interval <interval>	Seconds between file processing cycles. (Required)
-scada <scada>	SCADA source name. (Required)
-delimiter <char>	Allows a project to override the input file delimiter character with the specified character.
-rtdbs	Instructs RTAdapter to use a dedicated DBService instance . This is recommended if you are using the -dir RDBMS option to periodically scan scada_digital_in and scada_analog_in tables. To use this option you must have a DBService instance configured with a process name of rtdbs – see note above on RTDBService.

Software Configuration

Overview

The interface comes with a configuration support tool (rtipop) that can be used to help populate the standard SCADA configuration tables for incoming SCADA data (analog_measurements and digital_measurements). The rtipop process can be used to provide initial population of these tables.

Adapter Configuration

RDBMS configuration:

Tables involved:

- **digital_measurements:** standard SCADA configuration table for incoming digital data
- **analog_measurements:** standard SCADA configuration table
- **scada_ids:** RTAdapter/rtipop SCADA definition table - required for both.
- **scada_states:** RTAdapter string state to integer mapping - required.
- **scada_synonyms:** RTAdapter scada data attribute value mapping - required.
- **scada_analog_in:** RTAdapter polling table can be used to queue incoming analog SCADA updates - optional.

  Note: The RTAdapter process does not require any population into the scada_analog_in table. The use of scada_digital_in is entirely optional but is intended to be used in conjunction with scada_digital_in.
- **scada_digital_in:** RTAdapter polling table can be used to queue incoming digital SCADA updates - optional.

  Note: The RTAdapter process does not require any population into the scada_digital_in table. The use of scada_digital_in is entirely optional but is intended to be used in conjunction with scada_analog_in.
- **scada_points**: optional rtipop staging table - to help populate analog/digital measurements

  Note: The RTAdapter process does not require any population into the scada_points table. The use of the scada_points as a staging table is merely one solution for the ultimate problem of configuring the required SCADA interface tables (digital_measurements and analog_measurements). The use of rtipop is entirely optional.

To configure the standard SCADA input RDBMS tables (analog_measurements and digital_measuremens) using rtipop you might follow these steps:

1. Specify which devices have SCADA (via scada_points table and rtipop):

   Two options:

   1. The scada_points table is generally populated via attribute population during model build construction/update but can be populated after the fact by a custom (project specific) process.
   2. Populate **scada_points** RDBMS table via model build device attribute configuration.

      This option involves populating two attributes in the scada_points table:

      - **scada_name**: the name of the SCADA, as defined in scada_ids.scada_name (for example, USA).

- **rtu_alias**: SCADA unique identifier for reporting field device.

  The rtu_alias must only be unique within a particular SCADA (scada_name). An individual rtu_alias may well report multiple analog values (AMP, VOLT, KVAR, etc.) as well as digital and/or status values.

To set up the scada_points table as a standard Network Management System attribute table generally involves the following RDBMS tables:

- **device_attributes**: generic model build attribute configuration.
- **scada_points**: SCADA project specific attribute table.

Once the scada_points table is populated, the rtipop program can be used to expand the information in the scada_points attribute table to fully populate the more generic analog_measurements and digital_measurements tables (see notes below for how to use rtipop).

Note: A given “field device” corresponds to a given scada_points.rtu_alias and would typically be a breaker of some kind (often reporting both device status and multiple analog values). It could also be a transformer reporting analog values with or without status.

Below are two example device_attributes table entries to support population of the scada_points table via standard model build attribute population. For more information on this process, please consult the Network Management System Model Build process documentation. These are examples only.

```
INSERT INTO device_attributes (
        DEVICE_CLS,
        ATTR_NAME,
        TABLE_NAME,
        COLUMN_NAME,
        ATTR_TYPE,
        LENGTH,
        REQUIRED,
        MAINTENANCE
) VALUES (
        143, 'Rtu_Id', 'scada_points', 'rtu_alias', 3,
        32, 'N', 'Y');
COMMIT WORK;
       INSERT INTO device_attributes
        DEVICE_CLS,
        ATTR_NAME,
        TABLE_NAME,
        COLUMN_NAME,
        ATTR_TYPE,
        LENGTH,
        REQUIRED,
        MAINTENANCE
```

```
) VALUES (

        143, 'Rtu_Desc', 'scada_points', 'scada_name', 3,
        32, 'N', 'Y');

COMMIT WORK;
```

Example data field explanation:

143	Class of device which may report SCADA data. + project specific
scada_points	Attribute table to populate.
Rtu_Id	Attribute id as appears in *.mb file for device.
	+ project specific - SCADA device id (rtu_alias).
rtu_alias	scada_points column to populate with Rtu_Id value.
Rtu_Desc	Attribute id as appears in *.mb file for device. + project specific - SCADA system name (scada_name).
scada_name	scada_points column to populate with Rtu_Desc value.
3	Data type of string (ASCII field); always a 3 for a string
32	Maximum length of this attribute string (bytes) + project specific per scada_id len for SCADA.
'N'	Required attribute. + project specific – generally (N)o.
'Y'	Set to Y for model builder maintenance. Set this to "Y" if you want the Model Builder to maintain this table via the incremental model build process.

Once the scada_points table is populated the rtipop program can be used to expand this information to fully populate the more generic (required) analog_measurements and digital_measurements tables.

Run rtipop -h to get command line options. In general:

- rtipop [-debug [n]] -partition <n> -initFile <file>
- debug <n> - Turns debug on <to level n>
- partition n - Populate partition n (0 = all partitions)
- initFile - file - rti.dat initialization file (see below)

The rti.dat file is the configuration file used by the rtipop program. Based on data in this file, and entries in the scada_points table, rtipop populates the standard SCADA configuration (RDBMS) tables:

- digital_measurements
- analog_measurements

The scada_points table contains a record (row) for each device in the Network Management System model that has SCADA information associated with it. Each record has a "scada_name" column which, in order to populate one of the measurements tables, must match a "SCADA_Name" keyword in this configuration file. Where there is a match a row is populated in the appropriate (digital or analog) measurements table for each defined attribute.

Each defined Digital attribute for a given SCADA_Name populates an entry in the digital_measurements table.

Each defined Analog attribute for a given SCADA_Name populates an entry in the analog_measurements table.

The syntax rules for the rti.dat file are:

Lines with a leading # are treated as comments (ignored).

Leading blank space is ignored.

Only the first two non-blank tokens on a line are recognized.
The remaining tokens are treated as comments (ignored).

Blank lines are okay.

Attributes are associated to last defined SCADA_Name.

Keywords(they must match EXACTLY):

- SCADA_Name:
- Analog:
- Digital:

Note: The colon ":" character is a keyword delimiter. The colon must appear as the first character after the keyword in order for the keyword to be recognized.

Example rti.dat file:

```
SCADA_Name: USA
Digital: status    (Switch Position or "Status")
Analog: Amps_A
Analog: Amps_B
Analog: Amps_C
Analog: Volts_A
Analog: Volts_B
Analog: Volts_C
```

Example rtipop commands:

```
rtipop -partition 0 -initFile ${CES_DATA_FILES}/OPAL_rti.dat
rtipop -partition 3111 -initFile rti.dat -debug
```

3. Populate required measurement tables via device driven attribute population directly.

   This option bypasses the scada_points table, rti.dat and rtipop.

   Depending on what information is available in the supporting GIS this may or may not be practical. In general this option requires the GIS to have near complete information about what SCADA (attributes) are available for each model device with SCADA.

Other RTAdapter-specific RDBMS Population

scada_ids: RTAdapter name to unique integer value mapping.

- Generally used to SCADA name to a unique integer id.
- $NMS_HOME/sql/OPAL_scada_states.sql

scada_states: RTAdapter string to integer mapping.

- Generally used to map topology state and/or quality code information
- $NMS_HOME/sql/OPAL_scada_states.sql

scada_synonyms: RTAdapter scada data attribute/synonym value mapping.

- Generally used to map SCADA reported attributes to Network Management System attributes.
- $NMS_HOME/sql/OPAL_scada_synonyms.sql

Sample RTAdapter Configuration/Execution Sequence

To get RTAdapter up and running, the following general steps should suffice.

Start from the home directory for RTAdapter:

1. Login to Network Management System admin account with standard OPAL model configured and running.
2. Create RTAdapter specific RDBMS tables:

   ISQL.ces product_schema_scada.sql

   - creates scada_ids table
   - creates scada_states table
   - creates scada_synonyms table

   ISQL.ces <sql/OPAL_scada_points.sql

   - creates scada_points table
3. Create basic example configuration (change at will...)

   ISQL.ces OPAL_scada_ids.sql

   ISQL.ces OPAL_scada_states.sql

   ISQL.ces OPAL_scada_synonyms.sql

   ISQL.ces OPAL_scada_points.sql

   - Sample population data for scada_points table for OPAL model.
   - For example only. Need to hand edit to match your model objects.
4. rtipop -partition -initFile $CES_DATA_FILES/OPAL_rti.dat
   - This should populate **analog_measurements** and **digital_measurements** tables.
   - Make sure you have entries in the analog_measurements and/or digital_measurements tables when you are done with this step.
5. Validate the RTAdapter is in the $NMS_HOME/etc/system.dat file (see directions above).
   - Recommend using *-watch* and possibly the *-debug* option to start; helps to identify configuration issues.
6. If the system.dat file is using the *$NMS_SCADA_SCAN_FILE_DIR* environment variable to specify the SCADA scan file directory, make sure this environment variable points to a directory that the RTAdapter process can both read and write. Generally, this means a

directory owned by the id that is executing RTAdapter. For example, *mkdir ~/usa*. At the same time, suggest creating a test data holding directory (for example, *mkdir ~/usa/tst)*.

7. Stop and restart Oracle Utilities Network Management System services (sms_start.ces).
   - Make sure RTAdapter is running.
8. The $NMS_HOME/templates/rtiadapter.dat.template file contains sample RTAdapter incoming data blocks. You can use the example data blocks in this file to validate basic RTAdapter functionality.

For example:

Copy example data blocks from the rtiadapter.dat.template to individual test files under ~/usa/tst (using the example configuration above); cut the following out of rtiadapter.dat.template SCADA data file to "live" RTAdapter scan file directory to test.

1. Copy the following lines into a file - say BR2413_open

   ```
   DATA
   OBJECT|BR2414
   BREAKER_POS|OPEN
   END_DATA
   ```

2. Copy the following lines into a file - say BR2413_close

   ```
   DATA
   OBJECT|BR2414
   BREAKER_POS|CLOSED
   END_DATA
   ```

3. Copy BR2413_open and BR2413_close to ~/usa/tst (following example above).
4. cd ~/usa/tst
5. cp BR2413_open ..

   This should cause the BR2413 file to be read and processed by RTAdapter - you should see the BR2413 device open in the standard OPAL model.

6. cp BR2413_close ..

   This should cause the BR2413 file to be read and processed by RTAdapter - you should see the BR2413 device close in the standard OPAP model.

7. Follow other examples for conditions and quality codes.

   Turn debug on RTAdapter to see what is going on. You should be able to send RTAdapter debug messages on the fly:

   ```
   Action any.USA debug on
   ```

8. Validate that devices are changing state in the Network Management System viewer as you execute steps 5 and 6 above ("cp BR2413_open .." followed by "cp BR2413_close ..") sequence over and over.

Sample RTAdapter polling configuration/execution sequence

To get RTAdapter up and running, the following general steps should suffice.

Start from the home directory for RTAdapter:

1. Login to Network Management System admin account with standard OPAL model configured and running.
2. Create RTAdapter specific RDBMS tables:

   `ISQL.ces ces_schema_scada.sql`

   - creates scada_ids table
   - creates scada_states table
   - creates scada_synonyms table

     `ISQL.ces ces_retain_scada.sql`

   - creates scada_digital_in and scada_analog_in tables
3. Create basic example configuration (change at will...)

   ```
   ISQL.ces OPAL_scada_ids.sql
   ISQL.ces OPAL_scada_states.sql
   ISQL.ces OPAL_scada_synonyms.sql
   ```
4. Validate the RTAdapter is in the $NMS_HOME/etc/system.dat file (see directions above).
   - Verify –dir is set to 'RDBMS'.
   - Recommend using –watch and possibly the -debug option to start – to help identify configuration issues.
5. Stop and restart Oracle Utilities Network Management System services (sms_start.ces).
   - Make sure RTAdapter is running.
6. Insert row into SCADA_DIGITAL_IN table either using alias or h_cls and h_idx with status = 'N'. The primary key on the scada_digital_in table is the id column – which is generally populated by a trigger on the scada_digital_in table that fires on insert and populates the id column with the next value in a sequence.
   - Example sql statement:

     ```
     INSERT into scada_digital_in (
        h_cls,
        h_idx,
        alias,
        attribute,
        phases
        operation,
        quality
        source,
        status
     ) VALUES (
        206,
        1015054348,
     ```

```
       '',
      '0',
      '7',
      '1',',
      'SCADA',
      'N'
   );
   COMMIT WORK
```

Data Flows

SCADA Entry

The SCADA system sends fixed format files to the RTAdapter. The following format rules generally apply:

1. Actual SCADA data appears between ^DATA (the string DATA at the start of a line) and ^END_DATA.
2. Records between DATA and END_DATA are identified by OBJECT which must match an analog_measurements.rti_alias or digital_measurements.rti_alias entry. Where both the rti_alias matches the field after OBJECT| and the subsequent attribute matches the analog_measurments.attribute or digital_measurements.attribute field an update may occur (assuming new data is provided).
3. SYNCHRONIZE|TRUE is a special case used to synchronize conditions and is outside the standard DATA/END_DATA block. If set the line following SYNCHRONIZE|TRUE should be something like TYPE|note - to indicate the data that follows is to be used to synchronize “note” class conditions. For SYNCHRONIZE scan files the condition action code should be "syn" - not "add" or "rem".
4. All other fields are generally ignored.
5. For digital_measurements: device status: open or closed, battery low, etc. Example:

   ```
   DATA
   OBJECT|BR2414
   BREAKER_POS|OPEN
   END_DATA
   ```

6. For analog measurements: Amps (Amps_A, Amps_B, Amps_C):

   ```
   DATA
   OBJECT|BR2413
   AMPS_A|1.1
   AMPS_B|1.2
   AMPS_C|1.3|SUSPECT
   OBJECT|BR2414
   AMPS_A|2.1:4096
   AMPS_B|2.2
   AMPS_C|2.3
   END_DATA
   ```

   Both digital and analog measurements can include quality codes for each attribute. Quality codes are part of the standard Oracle Utilities Network Management System attribute definition and are contained within a 32-bit integer field. Bits 0->10 are reserved for Oracle Utilities Network Management System purposes. Bits 11->31 are available for project specific configuration. Quality codes are generally defined in the quality_codes configuration table. In the analog example above (AMPS_C|1.3|SUSPECT) the SUSPECT string must be defined in the scada_states table and map to a valid quality code integer. Analog entry with a quality code. Integers can also be used directly to provide quality codes (AMPS_A|2.1:4096).

7. Generic SCADA conditions (generally notes or tags - could be any condition) are also supported. To send a condition something like the following would be required:

```
DATA
OBJECT|BR2414
NOTE_0|add|WHO=system|TIM=2009-02-27T16:22:17|TXT=NOTE_0
txt|EXT=BR2414-note_0
END_DATA
```

The above text would "add" a note condition to the model on the device mapped to BR2414. The following keyword phrases can be used to specify common condition fields:

WHO= (who should be recorded as the creator of the condition - must be a valid NMS user name).

TIM= ISO timestamp for when the condition was added. Timestamp format must be defined in your $DATEMSK file.

TXT= Text string for the condition.text field (notes.text. tags.text, etc).

EXT= SCADA unique identifier for the created condition. This field is necessary to allow the external system to later remove the condition.

8. To remote the SCADA condition above:

```
DATA
OBJECT|BR2414
NOTE_0|rem|WHO=system|TIM=2009-10-27T16:22:17|EXT=BR2414-note_0
END_DATA
```

For potential use with the NMS MultiSpeak (other other Java based) SCADA adapters.

If desired the generic SCADA adapter can be used in conjunction with the NMS MultiSpeak adapter. The intent is use the RTAdapter to provide a buffering mechanism for “noisy” SCADA systems that could potentially generate many periodic analog (or digital) updates. Using RTAdapter to capture and bundle incoming changes reduces the impact on the NMS CORBA Gateway and NMS CORBA publisher – when compared to contacting an internal service directly. Using RTAdapter with the “-dir RDBMS” option allows changes to be captured and sent in bulk to internal NMS Services.

If configured to do so the NMS Web Gateway APIs used by the MultiSpeak SCADA interface will write to the scada_digital_in and scada_analog_in tables when processing updates from an external SCADA. SCADA measurements submitted using Web Gateway APIs 'updateDigitalStatuses' and 'updateMeasurements' can be written into staging tables SCADA_DIGITAL_IN and SCADA_ANALOG_IN instead of being submitted directly to DDService. This behavior is controlled by three configuration properties, which can be added to the CentricityServer.properties file.

1. intersys.use_db_for_scada_statuses

   If set to 'true' than device status updates received from SCADA system will be written to the SCADA_IN database table.

2. intersys.use_db_for_scada_digitals

   If set to 'true' than updates to digital values received from SCADA system will be written to the SCADA_IN database table.

3. intersys.use_db_for_scada_analogs

   If set to 'true' than updates to analog values received from SCADA system will be written to the SCADA_ANALOG_IN database table

By default all the above properties are set to 'false', which means that SCADA measurements will be sent directly to the internal DDService process.

Information Model

Database Schema

SCADA_POINTS Database Table

The SCADA_POINTS table contains a row for each device in the Oracle Utilities Network Management System operations database that has SCADA information associated with it. Each record has a “scada_name” column which, in order to populate one of the measurements tables, must match a “SCADA_Name:” keyword in the rti.dat configuration file (see notes above for example rti.dat population). Where there is a match, a row is populated in the appropriate (digital or analog) measurements table for each defined attribute.

The SCADA_POINTS table is normally populated via device driven attribute population during the model build process. It is a staging table for the RTI population process. It is not accessed during adapter execution.

The schema for this table is defined in the file product_schema_scada_point.sql

SCADA_IDS Database Table

Column	Data Type	Description
ID	NUMBER	Numeric identifier for each “scada source” that we want RTI to process.
SCADA_NAME	VARCHAR(32)	Name for the scada source

The schema for this table is defined in product_schema_scada_id.sql file. The script, OPAL_scada_ids.sql, populates generic SCADA sources for the OPAL model. A source is any SCADA system that can provide information to the adapter. There could be one SCADA source defined for each of multiple SCADA vendors, or a utility may choose to divide their territory into multiple regions, with each region acting as a separate SCADA source. Each SCADA source must have a name as well as a unique integer ID.

SCADA_SYNONYMS Database Table

The SCADA_SYNONYMS table contains all the synonyms for attribute name or values (e.g., KV_3, AMP_A, and CLOSE) used by RTAdapter in processing flat files of SCADA data input.

Column	Data Type	Description
scada_name	VARCHAR2(32)	SCADA name - from scada_id name
keyword	VARCHAR2(32)	SCADA unique attribute keyword string from SCADA system. Generally maps to an NMS attribute name but this is not required unless the scada_synonyms.attribute_alias field is left blank. If the scada_synonyms.attribute_alias field is left blank than the scada_synonyms.keyword field must map to a valid NMS attribute name - from attributes.name (table.column). For conditions this is unique name used by external SCADA to identify the condition class (NOTE, TAG, etc). NMS condition class must be in scada_synonyms.attribute_alias.
value	VARCHAR2(32)	For digitals: Customer value associated with keyword that indicates digital state (OPEN, CLOSED). For analogs: This field could be null, but the value is part of the primary key – so set to the same value as the keyword. For conditions: Must be “add”, “rem” or “syn”. The “syn” value is used for synchronization requests.
process_type	VARCHAR2(5)	For Digitals - 'D' For Analog - 'A' For threshold VOLT processing – ‘A_VOL’ For threshold AMP processing – ‘A_AMP’ For Conditions – ‘C’
attribute_alias	VARCHAR2(20)	Attribute name from attributes table. For digitals: The only way to get a model object to change status is to set this value to 'Status'. All other values are for digital attributes. For analogs: This field is optional and can be set to ‘’ (empty string). If this value is ‘’, the scada_synonyms.keyword is used as the attribute name. For conditions: This field is the condition class name (tag, note, etc).
status_value	VARCHAR2(20)	Numeric or string from scada_states.alias table. For digital status: This field is generally set to DEVICE_CLOSE or DEVICE_OPEN For analogs: This field is NULL. For conditions: This field is either a numeric condition status or a string that maps to a numeric condition status via the scada_states table. If it is a string it MUST start with an alpha (non-numeric) character.

For each implementation, define the customer specific <project>_scada_synonyms.sql file to specify the required synonyms.

SCADA_STATES

This table exists to allow for entering a character string into the scada_states.status_value field instead of an integer. For example 'DEVICE_CLOSE' instead of 2.

Column	Data Type	Description
SCADA_NAME	VARCHAR2(32)	Name of scada from scada_ids.scada_name
ALIAS	VARCHAR2(32)	Alias to map to integer
VALUE	INTEGER	Integer value to map to

OPAL_scada_states.sql defines the commonly used entries.

SCADA_DIGITAL_IN

The scada_digital_in table can be used by RTAdapter to queue incoming digital SCADA updates. RTAdapter, if configured to do so, should periodically poll this table and check for unprocessed rows (status='N'). If unprocessed rows are found RTAdapter will attempt to update the model according to data provided. If an error occurs, the scada_digital_in.error_code and scada_digital_in.error_description columns should be populated with some indication of the problem - status will be set to 'E'. If the update is successful the row is deleted.

Note that the database sequence scada_digital_in_sequence must be set up properly to create the primary key (scada_digital_in.id) value on insert.

If the scada_digital_in .attribute is a numeric it must match a valid NMS attribute number (0 is topology status – for example). If the scada_digital_in.scada_digital_in.attribute and scada_digital_in.operation values must be properly defined in the scada_synonyms and scada_states tables.

Column	Data Type	Description
ID	VARCHAR2(32)	scada_digital_in sequence generated pk
H_CLS	NUMBER(38,0)	NMS class of device – can be null if alias is not null.
H_IDX	NUMBER(38,0)	NMS index of device – can be null if alias is not null.
ALIAS	VARCHAR2(128)	SCADA point alias - can be null if h_cls and h_idx are NOT null.
ATTRIBUTE	VARCHAR(32)	SCADA attribute. If numeric, it must match a valid NMS attribute number. If it is a string, it must map to a valid NMS attribute number via the scada_synonyms table.

Column	Data Type	Description
PHASES	VARCHAR(4)	Intended phases for operation. If numeric must be between 1 and 7 – where 1 is A and 7 is ABC. If a string must map to a valid numeric via the scada_states table.
OPERATION	VARCHAR(32)	Operation. If numeric and used for attribute 0 (topology status) it must be 1(open) or 1(close) and the phase attribute must be set to indicate which phases are intended to operate. If a string it must map to a valid code for the attribute involved via a combination of the scada_synonyms table and/or the scada_states table.
OPERATION_DATE	DATE	Operation. If numeric and used for attribute 0 (topology status) it must be 1(open) or 1(close) and the phase attribute must be set to indicate which phases are intended to operate. If a string it must map to a valid code for the attribute involved via a combination of the scada_synonyms table and/or the scada_states table.
OPERATION_COUNT	NUMBER(10)	How many operations have occurred since the last scan – for momentaries (not presently supported).
CAPTURE_DATE	DATE	When operation captured.
QUALITY	VARCHAR2(32)	Quality code for attribute – can be numeric or a string. Either way it must be properly configured in NMS and must ultimately translate to be greater than 0x7FF (2047) and less than or equal to 0xFFFF. All quality codes below 0x7FF are reserved for NMS.
SOURCE	VARCHAR(32)	source/user name
STATUS	VARCHAR2(1)	Status of request: N = New E = Error
ERROR_CODE	NUMBER(38,0)	Error code
ERROR_DESCRIPTION	VARCHAR(256)	Error code description.

SCADA_ANALOG_IN

The scada_analog_in table can be used by RTAdapter to queue incoming analog SCADA updates. RTAdapter, if configured to do so, should periodically poll this table and check for data that has changed since the last update of the scada_analog_in.capture_date column. If potential updates are found RTAdapter will attempt to update the model according to the data provided. If an error occurs an error is written to the RTAdapter log file. If the update is successful no changes are made to the scada_analog_in table. This is to support the idea of continuous update of the scada_analog_in table from an external entity. The scada_analog_in table can be updated many times between RTAdapter scans. RTAdapter will "harvest" whatever appears to have changed since the last scan. It is expected that some form of merge statement would be used to update the scada_analog_in table – inserting if a record does not exist and updating otherwise – which triggers an update on the capture_date column.

Either a valid NMS handle OR a unique combination of analog_measurements.rti_alias and analog_measurements.attribute must provided for each record in the scada_analog_in table. If a valid rti_alias is provided but no valid handle is available you must set h_cls=0 and h_idx=0 . This supports the scada_analog_in primary key and is a way of telling RTAdapter which scheme to use to find the appropriate analog to update.

Column	Data Type	Description
ID	VARCHAR2(32)	scada_digital_in_sequence generated pk
H_CLS	NUMBER(38,0)	NMS class of device
H_IDX	NUMBER(38,0)	NMS index of device
ALIAS	VARCHAR2(128)	SCADA point alias - can be null if h_cls and h_idx are NOT 0.
ATTRIBUTE	VARCHAR(16)	SCADA attribute. If it is numeric it must match a valid NMS attribute. If it is a string it must be defined in scada_synonyms and map to a avalid NMS attribute.
MEASUREMENT	NUMBER	Analog update value.
MEASUREMENT_DATE	DATE	When operation occurred in field – not presently used.
CAPTURE_DATE	DATE	When measurement captured – could be updated by trigger on table update. This is the how RTAdapter determines what to examine during periodic polls.

Column	Data Type	Description
QUALITY	VARCHAR2(32)	Quality code for attribute – can be numeric or a string. Either way it must be properly configured in NMS and must ultimately translate to be greater than 0x7FF (2047) and less than or equal to 0xFFFF. All quality codes below 0x7FF are reserved for NMS.
SOURCE	VARCHAR(32)	source/user name

ANALOG_MEASUREMENTS

Column	Data Type	Description
H_CLS	SMALLINT	Object handle
H_IDX	INTEGER	Object index
PARTITION	INTEGER	Object partition handle
ATTRIBUTE	SMALLINT	Data attribute index (from ATTRIBUTES table)
TTL	SMALLINT	Time-To-Live Value
LIMIT_GROUP_ID	INTEGER	Object limit group
RTI_ALIAS	VARCHAR2(128)	RTI device measurement name
SCADA_ID	INTEGER	SCADA source identifier - matches scada_ids.id
RTU_ID	VARCHAR2(32)	RTU IDID - unique name within SCADA system.
QUALITY	INTEGER	Quality code
VALUE	FLOAT	Manual Replace Value
UPDATE_FLAG	INTEGER	Manual Replace Flag
ICCP_OBJECT	VARCHAR2(32)	ICCP mms object name
DISPLAY_ID	VARCHAR2(64)	ID for display call up
CONTROLLABLE	VARCHAR2(1)	Is this row controllable
ACTIVE	VARCHAR2(1)	Is this row active
SOURCE	VARCHAR2(33)	Source of measurements
COMMENTS	VARCHAR2(512)	Comment associated with
OFF_NOMINAL_TIME	DATE	Time quality went off-nominal

DIGITAL_MEASUREMENTS

Column	Data Type	Description
H_CLS	SMALLINT	Object handle
H_IDX	INTEGER	Object index
PARTITION	INTEGER	Object partition handle
ATTRIBUTE	SMALLINT	Data attribute index (from ATTRIBUTES table)
TTL	SMALLINT	Time-To-Live Value
LIMIT_GROUP_ID	INTEGER	Object limit group
RTI_ALIAS	VARCHAR2(128)	RTI device measurement name
SCADA_ID	INTEGER	SCADA source identifier
RTU_ID	VARCHAR2(32)	RTU ID
QUALITY	INTEGER	Quality code
VALUE	FLOAT	Manual Replace Value
UPDATE_FLAG	INTEGER	Manual Replace Flag
ICCP_OBJECT	VARCHAR2(32)	ICCP mms object name
DISPLAY_ID	VARCHAR2(64)	ID for display call up
NORMAL_STATE	INTEGER	Normal state for measure
CONTROLLABLE	VARCHAR2(1)	Is this row controllable
ACTIVE	VARCHAR2(1)	Is this row active
SOURCE	VARCHAR2(33)	Source of measurements
COMMENTS	VARCHAR2(512)	Comment associated with
OFF_NOMINAL_TIME	DATE	Time quality went off-nominal

This section provides an overview of the logical information model supported by this interface. The key objects supported by this interface include:

- Customers, which are defined using accounts, service locations and meters. This model is based upon the MultiSpeak model. Typically, as a practical note, the custId identifier may in fact be the same as the account number. Some extensions to the MultiSpeak model are used as required to address issues that are otherwise not addressed by MultiSpeak. The support of a bulk load process that reads a XML file, with defined customers to create the model. This process (createSql) can be ran to generate the SQL to be run on the production servers, or can directly create the customers.
- Trouble calls are also referred to as incidents within Oracle Utilities Network Management System. An incident is typically related to a customer, who in turn is related to a device. In the absence of a correlation to a device, a trouble call is classified as a 'fuzzy' call, which differentiates it from a call that can be directly correlated to the electrical distribution network.

- Outages, which are a consequence of the correlation of incidents. Outages are one form of an event that is managed by JMService. Some events are non-outage events, such as power quality. The type of call that is provided can identify such non-outage and outage events. Each call needs to be identified with a trouble code, which will determine the type of call that JMService will generate with in Oracle Utilities Network Management System.
- Devices, which are part of the electrical distribution network. Customers, outages and conditions may have relationships to devices. Typically customers are related to transformer devices. Outages are typically related to switch, fuse or transformer devices.
- Conditions (which can be specialized within Oracle Utilities Network Management System for the management of information such as tags, notes, etc.)
- SQL queries, result sets and transactions.
- Customer disconnections and reconnections for indicating customers who have been purposely removed from Service by the utility.
- Crew Outage States that will identify outage states that change as a result of a crew action. For example a crew that has been dispatched, assigned, or suspended from outage work would correlate to an action that may trigger an outage state change in Oracle Utilities Network Management System.

The information described by these models is formatted using XML for the purposes of exchange through this interface. The following table describes tags that are used in the XML definitions, and how they relate to the information model within Oracle Utilities Network Management System. The corresponding types used in these models are I = Integer, S = String, T = TimeStamp, C = Single Character, and F = floating point.

Chapter 3

SmallWorld GIS Adapter

This chapter describes the Oracle developed Smallworld data adapter. Since both the Oracle Utilities Network Management System and the Smallworld are highly configurable, there are many configuration options for the model build interface, but this document focuses only on the Smallworld Data Adapter. A related document, "Oracle Utilities Network Management System Model Build Interface" exists to provide details of Smallworld - Oracle Utilities Network Management System interface.

Model Build Process Overview

GIS to Oracle Utilities Network Management System integration is a multi-step process that generates an operational topological representation of the existing GIS database for use by the Oracle Utilities Network Management System. This process is referred to as the model build interface. During the model build interface, a single spatial grouping of GIS objects (known as a partition) is processed by the Oracle Utilities Network Management System model build engine before incorporating the partition data into the OMS data model. An electrical circuit (feeder) or polygon feature, such as a service territory boundary, can be used to define a partition. The Oracle Utilities Network Management System model build engine processes partitions in three stages before committing it to the OMS operational data model: 1) GIS data extraction, 2) model pre-processing and 3) data import into Oracle Utilities Network Management System.

The Smallworld Data Adapter extracts geospatial data in Oracle's vendor-neutral model preprocessing file format (MP file). Geospatial data in Smallworld is stored in a number of geographic data and attribute data tables. The GIS stores and renders the geospatial data as a contiguous map. For performance and architectural reasons, Oracle Utilities Network Management System requires GIS data to be brought into the system as smaller chunks referred as partitions.

An electrical circuit (feeder) or polygon feature, such as a service territory boundary could be used to define a partition. Landbase features should be separated from electrical features because this data is inherently different, and because landbase data is typically more static.

Oracle's Smallworld Data Adapter is able to:

- Extract the data by predefined partitions.
- Extract the electric facility data by the feeder; i.e. when a feeder is extracted, all the devices that take part in forming the feeder (i.e. electrically connected) will be extracted in a single MP file.
- Extracts Landbase features by predefined polygon boundary in a single MP file.

In order to minimize the data extraction and model build data volume, the extract is capable of identify the changed partitions since the last data extract operation.

Adapter Overview

The Adapter makes use of Smallworld GIS metadata via a Magik Visual Basic .Net d developed application to access the Smallworld GIS features. It utilizes default values as set in an accompanying Microsoft Access database, which has the same name as the executable (GElec2SPL) and resides in the same subdirectory. This required database is the “application configuration” database.

Obtaining the Software

The adapter setup files are included in the Oracle Utilities Network Management Optional Windows Applications download zip file that can be obtained from the Oracle e*Delivery site. After downloading the zip file, find the appropriate Adapter zip file within the NMS_GIS_Adapters directory and unzip to a location on your Windows server

It can be installed and run on any valid Smallworld supported platform namely PC with Windows XP/2000 or Unix as the supported platforms.

There are two methods of invoking the Adapter:

- Non-interactively by a command line with parameters.
- Interactively with a GUI.

The program can be launched in non-GUI mode from a Smallworld MagikOS command prompt with command line parameters. The user can either use the defaults as set in the application reference database or override them with passed parameters. As such, the program can be scheduled for “silent” launch as a Windows scheduled task. Note because this type of launch is “silent” and does not make use of a GUI, feedback is usually not provided to the workstation screen unless an error occurs.

The program can be launched from the workstation desktop or pull down menu. Using the GUI, the user can interactively control what data is extracted, what extract method is to be used, the type of data to extract, and the name of the output directory. The user is also provided with a status area, which includes progress bars and messages for monitoring the progress of the overall work order and the specific partition currently being extracted.

The remainder of this section describes the capabilities of the Adapter, how to install and configure it, and how to run it.

Adapter Capabilities

The Adapter is a partition-based tool that extracts objects and features as selected by their geographic location (i.e. based on the feeder name or partition to which they are spatially coincident).

Optionally, the Adapter can be used to detect changes to a GIS database, thereby automatically flagging partitions for extraction. Automated change detection by the Adapter is based on Smallworld version difference tool modification log table where the Adapter attempts to identify changed feeders or partition polygons since the last successful extract (based on creation and modification dates). If located, it will highlight the last freeze version for use with the change detect tool. The change detect tool is an optional automated mechanism for queuing partitions that contain objects and features that have been either changed, added, or deleted since the last extract.

On extraction, the Adapter retrieves:

- Partition details.
- Features.
  - Feature Object class and identifier.

- Connectivity.
- Geometry ic (Graphic) Components including feature annotation (text) component.
- Retrieves rows and Attributes - logical and physical for all the non-graphic components of the feature including annotation.

The Adapter produces output in a work order folder containing files in Oracle Utilities Network Management System Model Preprocessor (“.mp”) format and a log file of informational, warning, and error messages.

The details and issues of these topics are discussed in the following sections.

Feature Extraction

The Adapter generates Add records for each feature. The Add feature records list the feature, its topology, Diagram records, and Attribute records.

Object Identifier

The Adapter uses the feature class name and feature ID to identify an object. For example, “Switch 24.” Note the feature ID is unique within the feature class pool.

Connectivity

The Adapter retrieves the topology associated with a geometry at the node and link level. Features can be extracted at a single link or multiple link level.

Geometric Information

The Adapter determines the geometry of the object and uses it for the coordinates of the object as well as for any Oracle Utilities Network Management System diagram objects generated from the object: scale, height, angle, etc. The geometric diagram line should specify the feature class name.

The coordinates will be extracted in the defined coordinate projection system.

Coded Value Look Up

In Smallworld enumerators (code values) are hidden from the user and no lookup is required.

Class Names Containing Space Characters

It is possible for GIS class user names to contain spaces in them (e.g., “Overhead Line”). However, the Oracle Utilities Network Management System model build tools do not recognize or accept class names with spaces, so a conversion must be made.

The accepted convention is that all spaces be substituted with underscores (e.g., “_”). Therefore, if source data class names have spaces in them, the final output model preprocessor (.mp) files will have underscores substituting the spaces. An example would be a class named “Overhead Line.” The ADD statement in the extract output file would show the name “Overhead_Line” instead.

Oracle Utilities Network Management System Partitions with Index Maps

The Smallworld GIS database is a continuous map. For performance reasons, Oracle Utilities Network Management System requires maps to be tiled, each tile corresponding to an Oracle Utilities Network Management System partition.

The electrical features are extracted by a feeder, i.e. all the electric devices taking part to form a feeder are extracted in single mp file. The adapter also extracts electrical device owner features such as structure and poles in the same mp file. The feeder name is used to name the extracted mp file.

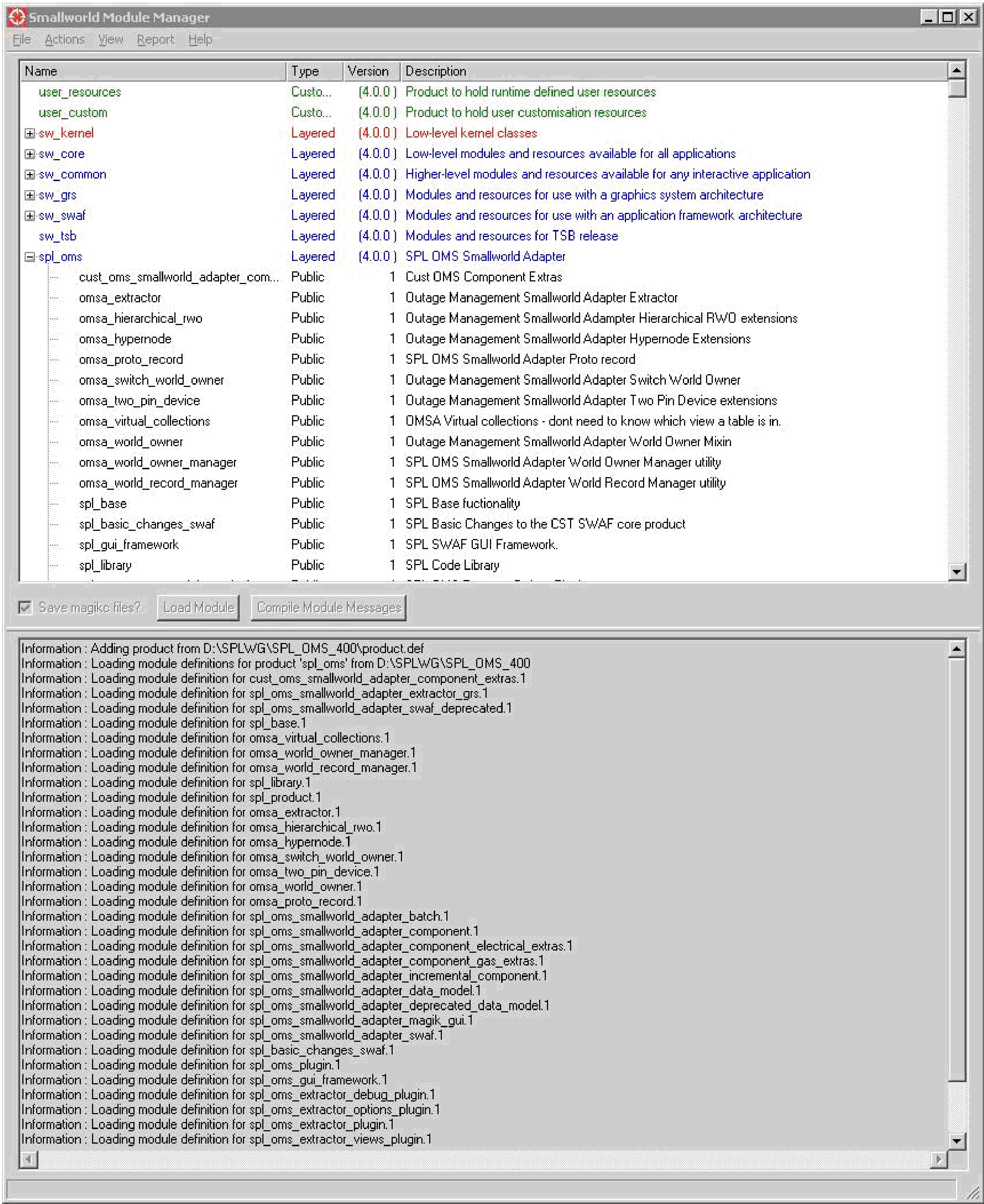
The spatial data extract requires a GIS polygon feature to extract the data. This polygon partitions can be any size and are not required to be uniform. The optimal sized partition polygon contains less than 5000 objects (devices, background text, and background graphics). For example, in dense urban areas of the model, smaller size partition polygon can be used. The adapter can use one of the existing polygon boundary class or users can define a new OMS specific polygon boundary class to define the partition polygon.

How to Build Smallworld Images

The product is supplied as compiled Magik components.

GE Smallworld CST 4.0 Systems

The Oracle Utilities Network Management System modules should be loaded as a separate layered product - SPL_OMS.



The key components, depending on whether electricity, gas or a combination of both is required, are:

- spl_oms_smallworld_adapter_component, spl_oms_smallworld_adapter_component_gas_extras and/or spl_oms_smallworld_adapter_component_electrical_extras ces_centricity_engine_gas_extras
- spl_oms_smallworld_adapter_batch
- spl_oms_smallworld_adapter_swaf and/or spl_oms_smallworld_adapter_grs

The dependencies contained in the module definition will ensure the correct loading of modules provided customer Smallworld images are built from software components.

A number of data model specific modules are provided for use with the numerous electrical starter kits. They include:

- omsa_hypernode
- omsa_two_pin_device
- omsa_hierarchical_rwo
- omsa_world_owner

The source code is not provided. All requests for access to the source for modification, bug fixes, etc. should be referred back to Oracle.

Setting up the Oracle Utilities Network Management System Smallworld Adapter

The following section describe how to set up the Oracle Utilities Network Management System Smallworld GIS adapter.

Map Tiles

Need to set-up the OMSA Map Tiles grids. These can be any size and shape. Ideally, the map sheets should be defined as a true square using conformal map projections. It would be normal to define the grid in the default coordinate system.

The tile size is defined to be the most efficient at the Oracle Utilities Network Management System model build and import stage. The impact on the size is fairly negligible on the Smallworld side unless the amount of objects in a tile is extremely large. A target of 3-5000 objects for data that is expected to be attributed with Oracle Utilities Network Management System, otherwise for “dumb” background a target of 10,000 is more appropriate.

For non-tile based extraction e.g., feeder omsa_map_tiles need to be created and match the feeder names

Sample grid code is supplied. Refer to ..\SPL_OMS_400\modules\lib\SPL\util\grids.magik

For simple testing purposes you can manually define a Map sheet.

Visibilities

The adapter requires that a definition of the objects to be extracted be defined. This consists of a number of aspects:

- Which fields are going to be exported for each object
- What is the prime geometry for an object that defines which tile the object is to be associated with. In Smallworld it is possible to define multiple geometries against the object, but typically only one is of importance. This is not important for non-tile based extraction.

Object Configuration

The following sections describe the individual objects that can be configured for Oracle Utilities Network Management System Smallworld GIS adapter.

Prime Location

The prime location defaults to centerline, then location. If neither of these geometries is suitable, a vector of suitable geometries should be defined called omsa_prime_object. Mapped geometries are taken into consideration if this feature is used.

E.g., hypernodes are already redefined with the Adapter

```
hypernode.define_shared_constant(:ces_prime_location,
 simple_vector.new_with(
     :pin1,
     :pin2
            ),
 _false)
```

OMSA Object ID

All objects that are exported are assumed to be uniquely identified by a field call omsa_object_id. This defaults to id field on each object, then rwo_id, then key_value

```
_method ds_record.omsa_object_id
            _if _self.responds_to?(:id)
                _then
       _return _self.id.write_string
                _else
       _return "????"
                _endif
_endmethod
```

If the default is inappropriate the object will require the behavior to be over written.

E.g., Poles are often numbered uniquely by a pole number, resulting in the behavior needing to be overwritten.

```
_method pole.omsa_object_id
                         ##
                         ##
              _return _self.pole_id.write_string
_endmethod
```

OMSA_internal_associated_objects

A customization hook is provided to enable objects that are related to an object within an internal world that exist within other worlds with no join fields available. A rope or stream of objects should be returned.

Dataless_rwo_joins

A customization hook to extract dataless rwos. A rope or stream of objects should be returned. E.g., used to extract hv_cable_annotation for an hv_cable.

OMSA_associated_objects

A customization hook to extract related objects. A rope or stream of objects should be returned.

Non_joined_connected_objects

A customization hook to extract topologically connected objects. A rope or stream of objects should be returned. Used to get joints or hypernodes etc. when not joined to a feeder or feeder object.

OMSA_additional_info

A customization hook to extract additional information or attributes. Additional attributes can be exported by:

```
a_omsa_engine.write_extract_attribute(<a_field_name>,<a_value>)
```

It is advisable that is only additional info is to be extracted consideration should be given to just adding some logical or Magik fields, and just adding the new “fields” to the configuration files.

OMSA_extract_world

Customization hook to control when “internals” are extracted.

Temporary Object

The following section describes the configuration of an Oracle Utilities Network Management System GIS adapter temporary object.

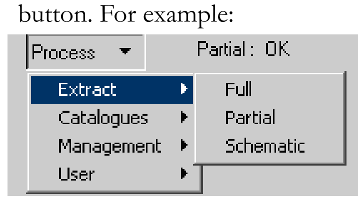
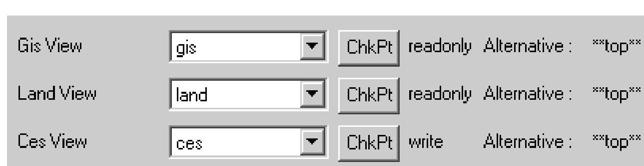
OMSA_proto_record

This object needs to used if transient objects need to be created for extraction.

The best example of this is to generate a pseudo road intersection object from a road centerline object.

Parameter Configuration

The following table describes Oracle Utilities Network Management System GIS adapter SPL_OMS_gui parameters.

Value	Default	Description
SPL_OMS_gui.process_menu		Rope defining the menu items to appear under process button. For example: 
SPL_OMS_gui.view_vector	{:gis,:ces}	List of what views to display on centricity_gui panel
SPL_OMS _gui.show_views?	_true	List Smallworld partitions on centricity_gui panel. 

Value	Default	Description
SPL_OMS_gui.debug?	_false	Enables debug logging to be enabled and controls. Also visible on centricity_gui panel. 
SPL_OMS_gui.toolbar?	_true	Places a toolbar at the top of centricity_gui panel.

OMSA_component Parameters

The following table describes Oracle Utilities Network Management System GIS adapter OMSA_component parameters.

Value	Default	Description
oms_component()	oms_component	Procedure to indicate which component to use
omsa_component.attribute_name_lowercase?	_true	Force attribute names in MP files to be lowercase
omsa_component.attribute_prefix	Omsa	Ensures compatibility with the prefixes used in earlier versions of the adapter
omsa_component.attribute_value_strip_unset	_true	Forces unset values to appear as “” instead of “unset” in MP file
omsa_component.cache_internal_worlds	_true	Caching ensures an internal world is on extracted once per run.
omsa_component.catalogue_view	Elec_view	The Smallworld view to find the catalogue tables
omsa_component.catalogues	{hv_cable_catalogue}	List of catalogue tables
omsa_component.customised_tile_names?	_false	Enables localized of MP files
omsa_component.distributor_join	_unset	
omsa_component.distributor_follow_joins	{}	
omsa_component.elec_view	Gis_program_manager. Cached_dataset(:gis)	Default electrical partition
omsa_component.feeder_collection	:feeder	Name of the feeder table
omsa_component.feeder_follow_isolatable_join?	_true	
omsa_component.feeder_follow_join	:isolatable_sections	
omsa_component.feeder_follow_joins	{}	
omsa_component.feeder_inline_world_joins	{}	
omsa_component.feeder_key	:id	Primary key of feeder table
omsa_component.feeder_world_joins	{}	

Value	Default	Description
omsa_component.include_hypernodes_with _parent	Equality_hash_table	Keys on omsa_map_class
omsa_component.include_hypernodes_with_parent?	_true	Gives control over how many MP files a hypernode can exist in.
omsa_component.include_world_owner_with_internals?	_false	e.g., Include substation details with the substation internals MP file.
omsa_component.isolatable_section_collection	:isolatable_section	Name of isolatable_section table
omsa_component.isolatable_section_follow_joins	{}	Join or pseudo joins to follow to pick up associated objects.
omsa_component.isolatable_section_inline_world_joins	{}	
omsa_component.isolatable_section_key	:id	Primary key of feeder table
omsa_component.siolatable_section_world_joins	{}	
omsa_component.land_view	Gis_program_manager. Cached_dataset(:land)	Default land / cadastre partition
omsa_component.log_full_object_counts?	_false	
omsa_component.omsa_prime_location_hierarchy	:centreline, :location, :coverage, :boundary, :annotation	If an object does not have a ces_prime_location defined, the order described is used.
omsa_component.report_attribute_errors?	_false	Useful to turn on to test adapter configuration files
omsa_component.restrict_world?	_false	
omsa_component.skipped_derived_fields?	_false	Enables derived fields e.g., SAP fields to be ignored
omsa_component.st_feeder_collection	:xxx	Name of the subtransmission feeder table
omsa_component.st_feeder_follow_joins	{}	
omsa_component.st_feeder_key	:st_feeder	Primary key of subtransmission feeder table
omsa_component.st_feeder_inline_world_joins	{}	
omsa_component.view_for_map_class()	:gis	Configure which partition to use for extraction class
omsa_component.update_tile_extract_details?	_true	Add counts and date/time to omsa_map_tile when extracted or marked for incremental
omsa_component.world_for_type	_unset	Default world for map type – used when schematics are in separate world.

Other Parameters

The following table describes Oracle Utilities Network Management System GIS adapter other parameters available for configuration.

Value	Default	Description
sector_rope.omsa_reproject()	_self	Enables local map projections to be applied to all sectors exported with the adapter
omsa_map_tile.omsa_tile_name	:known_as	Customizable tile name
omsa_extract_parameter.type_abbreviation	Cadastral - CAD Feeder - FDR Isolatable section - IS LV feeder - LVF Subtransmission Feeder - STF Schematic - SCH Schematic - background - SCB Schematic - dxf - SCD	Equality_hash_table of abbreviations to be used in file names

Incremental Adapter Parameters

The following table describes Oracle Utilities Network Management System GIS adapter incremental parameters available for configuration.

Value	Default	Description
omsa_incremental_component. checkpoint_name	{OMSA_<type>}	Checkpoint name. Each map_class has a separate checkpoint.
omsa_incremental_ component omsa_alternative	|OMSA	Alternative that ALL changes are checked in.
omsa_incremental_ component. additional_date_checkpoint?	_false	Places and additional date checkpoint along with standard checkpoint_name.
omsa_incremental_component. csv_report?	_false	Additional one line per object incremental report.
omsa_incremental_ component omsa_incremental_ csv_heading	Object,id,tiles	Heading for CSV report
<object>.incremental_csv_string		Customization hook for csv report.

Configuration Files

Adapter configuration files need to defined to determine which objects and fields are to be exported.

The format is <object_name>,field1,field2

E.g., substation,known_as,type,pole_mounted?,date_built,location,annotation

In most cases the main electrical will be defined in MAP.MPC with the Smallworld internals being defined in internals_schematic.MPC

A utility has been provided to quickly generate some configuration files. The utilities set up the object list based on the Smallworld visibilities.

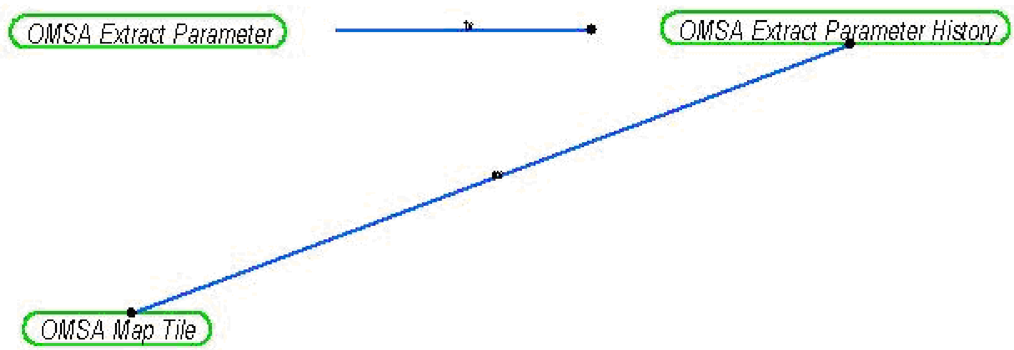
E.g., for the Smallworld Electrical demo or Smallworld Electrical Starter Kit the following method will generate the configuration files.

```
_method gis_ds_view.create_elec_demo_base_configuration(a_file_name)
   ##
   ##
   v.create_omsa_config_file(a_file_name,
        {:auxiliary_cable,
         :distribution_pillar,
         :hv_cable,
         :hv_line,
         :hv_joint,
         :lv_joint,
         :link_box,
         :lv_cable,
         :lv_line,
         :pole,
         :substation,
         :tower,
         :supply_point,
         :hypernode
        })

_endmethod
$
```

Data Model

The OMSA partition contains the map tile definitions and extraction parameters.



Execution

The following sections describe the modes of execution available for Oracle Utilities Network Management System GIS adapter.

Adapter Execution

The Adapter can be run either in the background with command line parameters or interactively with a GUI.

The Adapter supports two formats:

- Classic - required form Unix execution and Smallworld CST releases prior to 4.0
- Application Framework - commonly referred to as SWAF

Launching Adapter Interactively

The Smallworld Data Adapter can be launched from a Smallworld Application Menu or Toolbar. After a few moments, the program will then present a Smallworld Adapter dialog box.

Batch Execution

A batch method has been provided to enable extraction to be performed without a graphics system. If images are built correctly it should also be possible to build an Oracle Utilities Network Management System Extraction image that does extraction upon image startup.

How to run from the Magik prompt?

```
Batch_oms << spl_omsa_batch.new()
Batch_oms.batch_process(:batch_run)
```

The last statement executes the method batch_run in a separate thread.

Support is provided for the following extraction types

- extract_full()
- extract_partial()
- extract_schematic()
- incremental()

A sample setup is provided for spl_omsa_batch.batch_run

For example, to run a feeder extraction:

```
_method spl_omsa_batch.batch_run
       ##
       ##
       # set alternative paths
       _self.incremental("feeder")
       _self.extract_partial("feeder")
_endmethod
```

Consideration should be given to ensure the correct alternatives are used for extraction.

If multiple copies of the adapter are required for extraction the following slots can be set:

- Spl_omsa_batch.number_of_extractors << n
- SPL_omsa_batch.extractor_number << n

The extractor number should be in the range 0-(number of extractors -1)

Topology

Full Smallworld topology is maintained when data is exported. The topology is exported at the node and link level. It is normally only Smallworld developers that would work at this level. Most customers are unaware of the topology at the base level.

This means that split_link topology is handled correctly upon export.

For example, an hv_cable is exported as a series of multiple links, or more that two ports

There may be some customization required in this area. E.g., sometimes a pole-mounted switch is represented as a single point in the geographic, thus not using the standard two_pin_device behavior. This can be handled by overriding the method OMSA_write_port_info. An example of this is listed below:

```
_pragma(classify_level=basic, topic={ces,extraction},
usage={internal})
_method switch.OMSA_write_port_info(extractor)
    ##
    ##
    if self.geolocation _isnt _unset
    _then # internal 2 pin switch
        _super.OMSA_write_port_info(extractor) # use std two_pin_mixin
behavior
    _else
                    # geo switch
        extractor.write_extract("   PORT_0=", self.geolocation.node_id,
";", newline_char)
        extractor.write_extract("   PORT_1=", self.geolocation.node_id,
";", newline_char)
    _endif
_endmethod
```

Output of Adapter

The model preprocessor import files (.mp), the log file, and the work order time stamp file are all placed in a folder with the work order name. This folder will appear in the output directory as designated in the “Defaults” table. If Change Detection was run, a log file of the process will be placed in the root of the output directory.

Note that if the extract is still in progress or does not complete, the work order time stamp file will not appear in the output directory. The work order time stamp file will only appear if the complete extract finishes successfully. As such, external processes can use its presence as an indicator of a completed extract.

Performance

Performance of the extraction is currently at “GIS speeds." The non-spatial data extract (i.e. Electric feeders) does not depend upon the GIS Spatial Scanning ' AreaReportUtility application to locate the features located within a polygon partition so extracts is faster compare to spatial data extract. The spatial data extract uses Smallworld's AreaReportUtility application to identify features located within a polygon area.

The number of class components being extracted will dramatically affect Adapter performance. Therefore, the ExcludeFeature and ExcludeComponent table should be populated. The ExclueComponent table ensures that feature information will be extracted but components listed in the ExcludeComponent table will not get extracted. For example detail document, raster, or hyperlink document components are not required for the OMS. These components should be listed in the ExcludeComponent table.

The time required to perform change detection will vary and is dependent on the number of features that have changed since the last extract. The list of changed feature does not provide the spatial information so Adapter has to depend upon AreaReportUtility to identify the spatial polygon of the changed feature.

Data Model

The following illustrates a CASE tool descriptive report for Oracle Utilities Network Management System GIS adapter.

```
CASE Tool Descriptive Report



Objects:

    omsa_extract_parameter (OMSA Extract Parameter )
    omsa_extract_parameter_history (OMSA Extract Parameter History)
    omsa_map_tile (OMSA Map Tile )

Object name: omsa_extract_parameter ( OMSA Extract Parameter )

Properties:

    Record exemplar:             omsa_extract_parameter
    Visible fields ( default ):
                                 type
                                 last_extract
                                 projection
                                 output_directory
                                 log_file_directory
                                 configuration_directory
                                 incremental_directory
                                 report_directory
                                 omsa_extract_parameter_historys

Field summary:

       Physical fields          Field type           Mandatory  Default value  Unset value
    *  id                        sys_id               True
```

```
     V type                      ces_map_tile_type     True       "unknown"
     V projection                ds_charci_vec ( 32 )  False                     ""
     V configuration_directory  ds_charci_vec ( 60 )  False                     ""
     V output_directory          ds_charci_vec ( 60 )  True
     V log_file_directory        ds_charci_vec ( 60 )  True
     V incremental_directory     ds_charci_vec ( 60 )  False                     ""
     V report_directory          ds_charci_vec ( 60 )  False                     ""

    Logical fields  Field type
    last_extract    ds_charci_vec ( 50 )

    Join fields                      Relationship type  To object
    omsa_extract_parameter_historys  1:n                omsa_extract_parameter_history

Object name: omsa_extract_parameter_history ( OMSA Extract Parameter History )

Properties:

    Record exemplar:             omsa_extract_parameter_history
    Visible fields ( default ):
                                 omsa_extract_parameter
                                 extract_type
                                 type
                                 started
                                 completed
                                 database
                                 database_alternative
                                 database_checkpoint
                                 configuration_directory
                                 output_directory
                                 log_file_directory
                                 incremental_directory
                                 report_directory
                                 omsa_map_tiles

Field summary:

       Physical fields          Field type           Mandatory  Default value  Unset value
    *   id                        sys_id               True
     V  extract_type              ces_extract_type     True
     V  type                      ces_map_tile_type    True       "unknown"
     V  started                   ds_time              True                      date_time
     V  completed                 ds_time              False                     date_time
     V  database                  ds_charci_vec ( 32 )  True
     V  database_alternative      ds_charci_vec ( 32 )  True
     V  database_checkpoint       ds_charci_vec ( 32 )  False                     ""
     V  configuration_directory   ds_charci_vec ( 60 )  False                     ""
     V  output_directory          ds_charci_vec ( 60 )  True
     V  log_file_directory        ds_charci_vec ( 60 )  True
     V  incremental_directory     ds_charci_vec ( 60 )  False                     ""
     V  report_directory          ds_charci_vec ( 60 )  False                     ""
      I ces_extract_parameter_id  sys_id               True

    Join fields             Relationship type  To object
    omsa_map_tiles          m:n                omsa_map_tile
    omsa_extract_parameter  1:n                omsa_extract_parameter


Object name: omsa_map_tile ( OMSA Map Tile )

Properties:

    Record exemplar:             omsa_map_tile
    Visible fields ( default ):
                                 known_as
                                 name
                                 tile_name
                                 type
                                 extract?
                                 last_extract_count
                                 last_extract
                                 last_incremental
                                 omsa_extract_parameter_historys
                                 coverage
                                 annotation

Field summary:

       Physical fields     Field type           Mandatory  Default value  Unset value
```

```
*  id                  sys_id               True
 V known_as            ds_charci_vec ( 30 )  True
 V name                ds_charci_vec ( 30 )  False                      ""
 V type                ces_map_tile_type     True       "unknown"
 V extract?            ds_bool               True       _true
 V last_extract        ds_time               False                      date_time
 V last_incremental    ds_time               False                      date_time
 V last_extract_count  ds_uint               False                      0

Logical fields  Field type
tile_name       ds_charci_vec ( 30 )

Geometry fields                    Field type   Mandatory  Manifold
coverage                           simple_area  False
cadastral_coverage                 simple_area  False
diagram_coverage                   simple_area  False
dxf_coverage                       simple_area  False
feeder_coverage                    simple_area  False
gas_coverage                       simple_area  False
isolatable_section_coverage        simple_area  False
lv_feeder_coverage                 simple_area  False
map_coverage                       simple_area  False
overview_coverage                  simple_area  False
schematic_coverage                 simple_area  False
schematic_background_coverage      simple_area  False
schematic_dxf_coverage             simple_area  False
subtransmission_feeder_coverage    simple_area  False
annotation                         text         False
cadastral_annotation               text         False
diagram_annotation                 text         False
dxf_annotation                     text         False
feeder_annotation                  text         False
gas_annotation                     text         False
isolatable_section_annotation      text         False
lv_feeder_annotation               text         False
map_annotation                     text         False
overview_annotation                text         False
schematic_annotation               text         False
schematic_background_annotation    text         False
schematic_dxf_annotation           text         False
subtransmission_feeder_annotation  text         False


Join fields                      Relationship type  To object
Join fields                      Relationship type  To object
omsa_extract_parameter_historys  m:n                omsa_extract_parameter_history
```

Chapter 4

ESRI ArcGIS 9.x Adapter

This chapter describes the ESRI ArcGIS 9.x Adapter for Oracle Utilities Network Management System. It includes the following topics:

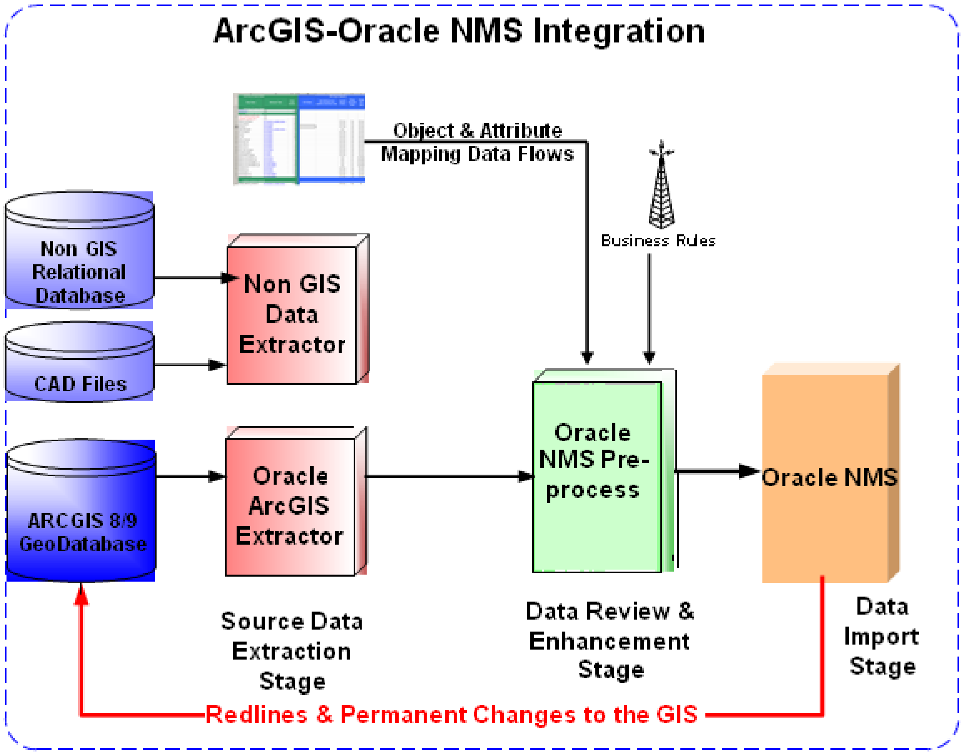
- **Adapter Overview**
- **Modeling Guidelines**
- **Adapter Functionality**
- **Installing ArcGIS to Oracle Utilities Network Management System Adapter**
- **Installing the Application**
- **Configuring the Adapter**
- **Freeze Versions Created and Used by the Adapter**
- **Running the Adapter**
- **Usage Scenarios**
- **Populating the Partition Listbox Using PartitionQueue Table**
- **Partition Loading**
- **Output of Adapter**
- **Performance**
- **Issues**

Adapter Overview

Oracle Utilities Network Management System model interface and the ArcGIS 9.x are highly configurable, there are many configuration options. This document includes a discussion of the choices and alternatives. A related document exists to provide project-specific configuration information.

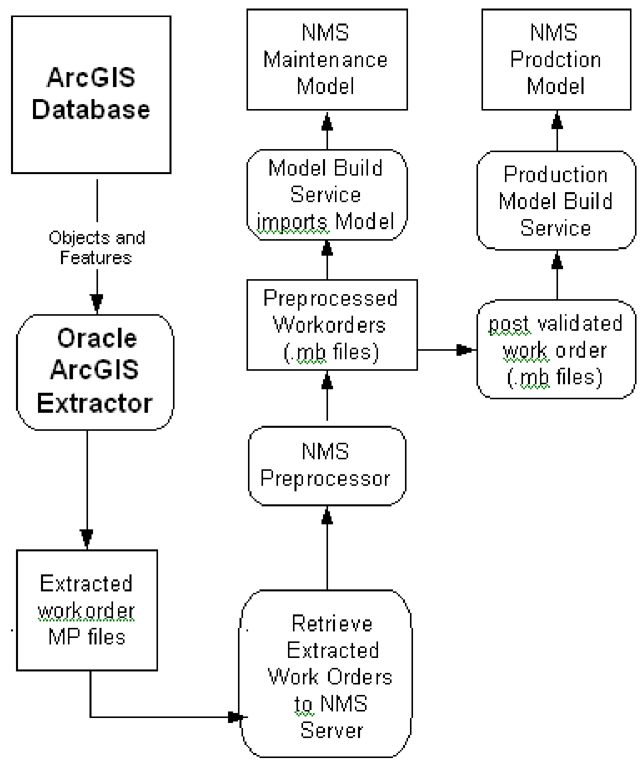
The Oracle Utilities Network Management System receives data from the ESRI ArcGIS 9.x to populate its topology model and provide graphic views of the electrical network and landbase. This is done as required using the ArcGIS:Oracle Utilities Network Management System interface to keep the Oracle Utilities Network Management System current with the network and landbase being maintained in the GIS.

The interface is one-way: updates from the GIS flow to Oracle Utilities Network Management System for processing. The feedback to the GIS system is typically done using red lining on a map.



Information Model

The following dataflow diagram illustrates the information flows and processes relevant to the model interface.



The GIS maintains several versions of the dataset. Drafters check out data versions to make changes typically related to a work order. When the changes are complete, the Drafter checks the version back in via reconciliation and posting tools. One specific version, typically “DEFAULT” is used by convention to maintain the “current” or “as-posted” data that is extracted and provided to the OMS.

Changes to the GIS are extracted in groups of related changes called an Extracted Work Order. The specification of changes to the GIS is done either manually or by using automatic change detection, which is discussed later in this document.

The extraction program is usually run after work order(s) with changed data have been posted back to the geodatabase. It can be run either manually (interactively) or scheduled to run periodically. The extraction runs on a PC-based ArcGIS workstation.

The Extracted Work Order is a folder containing the GIS extracted files in Model Preprocessor (.mp) format. There is one file per partition. The folder also contains:

- Control file. This is used later in the Oracle Utilities Network Management System work order processing. Its timestamp facilities the correct sequencing of work orders.
- Extract log file. The log file contains the detailed information about the entire data extract process such as extract process start and stop time, type and count of objects extracted and data errors

The default folder name contains a timestamp. However, the user can override it.

Once the Extracted Work Orders are complete, they can be moved to the Model Maintenance Server by the Retrieve Extracted Work Orders process. This is a script that uses a file transfer such as FTP. This script can be scheduled to run periodically such as daily.

On the Model Maintenance Server, the Preprocessor works on these files and generates Preprocessed Work Orders.

The Preprocessed Work Order is a folder containing the files in model build import format (.mb). Similar to the Extracted Work Order, there is one file per tile, control files, and log files. Again, the folder name should contain a timestamp to facilitate processing the extracts in the correct sequence. For convenience it is the same as the Extracted Work Order name.

The Model Build process on the Oracle Utilities Network Management System Model Maintenance Server builds the data model. Oracle Utilities Network Management System Model Validation tools verify the Model Build process results. The user can review the Model Build process log file to identify errors that could be corrected in the GIS or intermediate Model Preprocessor (MP) files. The process repeated until the work orders are correct.

Validated Work Orders are posted to the Production Server. This is a script that uses a file transfer such as FTP.

The Model Build process on the Production server builds the model. A model build is performed here so that any local edits, done by operators, are correctly merged in with the new work orders. After the build, the changes can either be accepted (committed) by the control room operator or rejected.

Further details on the building of models in Oracle Utilities Network Management System are not specific to the ArcGIS interface and are documented in Oracle Utilities Network Management System Model Guide.

Hardware and Software Environment

The ArcGIS 9.x processing for the data Adapter takes place on a Windows workstation, and optionally on an ArcSDE server.

The ArcGIS extraction workstation environment consists of:

Hardware platform	PC workstation – Recommended: Pentium III or better, 512MB Ram or more, Hard Disk Space Min: 1GB however must have enough beyond that for your GIS Extraction Files (.mp files) and extraction log files.
Operating system	Windows XP/Window Server 2003
Networking protocol	TCP/IP
Network media	Ethernet
Database product	Geodatabase via ArcSDE server, PersonalArcSDE, Geo File Database, or Geo Personal Database
Third Party Products	ArcGIS 9.x Microsoft Access for editing Personal Geodatabases (small test models only), and for editing the Adapter reference database.

Related Documents

The following sections recommend additional reading.

ArcGIS Documents

Modeling our World, Michael Zeiler, ESRI Press 1999. Overview, covers versioning and annotation.

Building a Geodatabase, Andrew MacDonald, ESRI Press 1999. User guide, covers versioning and annotation.

ArcObjects Developer's Guide, ESRI Press 2003. Guide to programmatically extending ArcGIS.

Object Model Documents

Implementing ArcFM Energy: Electric Utility Model Component Reference, Miner & Miner, 2000.

Modeling Guidelines

This sections describes how the ArcGIS geodatabase should be used to support the Oracle Utilities Network Management System model interface.

Supporting Oracle Utilities Network Management System Partitions with Index Maps

The ArcGIS geodatabase is a continuous map. For performance reasons, Oracle Utilities Network Management System requires maps to be tiled, each tile corresponding to an Oracle Utilities Network Management System partition.

The tiles can be any size and are not required to be uniform. The optimal sized tile contains less than 2500 objects (devices, background text, and background graphics). For example, in dense urban areas of the model, smaller tiles can be used. There is no danger in making the tiles too small, other than the increased overhead in maintenance.

A single tile class may be preferred, or several can be used so long as the same class is applied across the extract. Landbase features should be separated from electrical features since this data is inherently different, and because landbase data is typically more static.

To support tiling, index maps can be defined by drawing polygon features in a feature class within a dataset. Such feature classes might be named:

- ElectricTile
- LandbaseTile

A text attribute (e.g., "TileID") of the partition feature class will be used to store the unique name of each tile (e.g., "Downtown_NW", "Downtown_SE"). On extraction, the TileID becomes the name of the Oracle Utilities Network Management System partition.

Ideally, the partitions used for electrical data extraction should be drawn so they do not split any point features such as electrical switches. Note that splitting linear features such as conductors is acceptable.

Supporting Oracle Utilities Network Management System Symbology

Distinct symbols in Oracle Utilities Network Management System require distinct Oracle Utilities Network Management System classes. The model interface can generate an Oracle Utilities Network Management System class from a combination of GIS class and attributes. If a specific symbol is desired in Oracle Utilities Network Management System, then the GIS class and attributes must be present.

For example, the concrete GIS class "Transformer" has an attribute "Subtype" with values such as PowerTransformer, DistributionTransformer, etc. The model interface can use different symbols for PowerTransformer and DistributionTransformer by generating Oracle Utilities Network Management System classes PowerTransformer and DistributionTransformer based on the GIS class and attribute value.

Supporting Change Detection

Because of the size, it is usually not practical to extract the entire geodatabase each time a change is made. The goal of change detection is therefore to automatically queue partitions containing objects that have changed since the last successful extract. That way, only those partitions are extracted and passed through the model interface.

Change detection in the GIS is not intrinsically supported but the Oracle Utilities Network Management System model interface has functionality (Incremental Model Build) for handling changes.

The two options for change detection are manual and automatic.

There are several approaches available for automating change detection:

- Using database triggers to log the fact that a change has been made. This is considered unreliable because some changes are cached and don't make it into the more persistent tables until a database compress operation is done.
- Using a change-detection modification field on features.
- Read from an external source the partitions that contain changes.
- Comparing a static "last-extracted" database version to a current or "as-built" database version.

The third and fourth approaches above, namely reading from an external source the partitions that contain changes, and using ArcGIS database versioning and reconciliation tools for change detection are the implemented approaches. These approaches are discussed in greater detail in the sections that follow.

Manually Track Affected Tiles

This option requires keeping track of what changes have been made manually and then doing an extract on those specified tiles.

Automated Change Detection Using a Modification Field

A modification field on each feature could be used as a change detection field. Note that this option is not implemented. This is not a preferred option to track the changes.

One option is to examine the modification date on the version relationship to determine what has changed since the last extract. This assumes the modification date is adjusted to the time the posting took place rather than the time the edit occurred on the work order version. This examination has to look at every object in the geodatabase, which would be a time and process intensive operation. Also, there is no way of correlating these changes back to any meaningful reference such as a work order.

Another option is to configure the post operation to run the Adapter on any tiles affected by the work order version. This has the advantage of doing extracts against proposed work and later extracting "as-built" work.

In either case, the Adapter would perform a selection such as:

- Select all features with modification time greater than last extract. Alternatively, select all features with description equals the work order name. This selection may be further constrained to look for features within a political zone.
- Select all tile features containing the above selection set.
- Select all features contained in the selected tile features.

Then, utilize the Adapter to extract all the selected features.

This approach is likely to perform poorly. To optimize it, additional polygons could be defined. Polygons representing political boundaries and work regions and divisions of the utility corresponding to the data maintenance activity, could be applied to restrict model building to a specific area. For example, if the maintenance of the dataset is structured by Division, then a spatial query to determine changes after maintenance can be restricted to those features within that Division.

Automated Change Detection from an External Source

This is one of the implemented approaches. It assumes that change detection is performed external to the Adapter, and the results are provided to the Adapter in the form of a "list" of affected partitions.

To work, the external source of partitions containing changes must be a table in the same GIS database (geodatabase) as the data for extraction, and must follow a predefined schema. This table must also exhibit the same access rights and privileges for the database/SDE users as the other GIS model data does.

Once this table exists, external tools must be used to populate and manage the table contents. The Adapter has no control over this table, so it assumes the contents are correct and accurate. At present it does not write to or update this table, except to mark the extraction completed information.

The Adapter is configured to read from this table the change detect results, or the "partition sets" as defined by the external change detect tools.

Automated Change Detection using ArcGIS Versioning

This is the other implemented option. It uses the ArcGIS versioning and reconciliation tools in conjunction with some custom ArcObjects code to compare a last-extracted "freeze" version to the current or "proposed for OMS" version. The intent is to automate detecting data changes since the last extract.

To work, the GIS database must be registered as versioned. Note that version management can be performed through the ArcCatalog interface. A version can be considered a "copy" of the data that maintains state information and metadata such as the name, owner, description, creation date, modification date, parent version, and access permissions for the version.

To work properly, a "current" or "as-built" version of the data is required. This version is where all normal data changes or modifications should be posted, including work order completions. For simplicity, the system administrator can use the DEFAULT version or they may choose to create a new "current" or "as-built" version from the DEFAULT.

By design, every time an extract is successful, the Adapter creates a "freeze" version of the database. As a result, this historical image or benchmark of the data state is created that can be referred back to in the future.

Meanwhile, the change detect mechanism in the Adapter is designed to compare two versions of the database to determine the geographic location of changes to the database (i.e. objects that have been added, deleted, moved, or modified).

Together, the versioning tools and Adapter's change detect mechanism empower the Adapter to automatically detect changed features between the current database version and the freeze (created at the last extract). This information is then used to queue the appropriate partitions or tiles for subsequent extraction.

Then, when run, the Adapter extracts all features and objects located within the queued partitions. On completion of the extract, the "freeze" version is then made available for reference as the new data benchmark, for use in subsequent change detection when the Adapter is run next time.

ArcGIS Data Maintenance

Prior to launching the Adapter, a proper data maintenance environment with procedures and rules should be implemented. Requirements for setting up a proper maintenance environment are outside the scope of this document. These procedures will vary widely between installations, depending on factors such as whether the database is versioned. As a result, generalizations and assumptions have been made in this section.

Unversioned Database

If the database is not versioned, then maintenance will be simpler since any extract performed on an unversioned GIS database will therefore be of the default data.

Since there will only be one version, be aware that making changes to the GIS database should be scheduled with other users in order to avoid data sharing and locking conflicts. For similar reasons, extracts should also be scheduled with other database users. Typically, this could be accomplished by either running the Adapter after hours, or by performing extracts against a “mirror” database of the production GIS data.

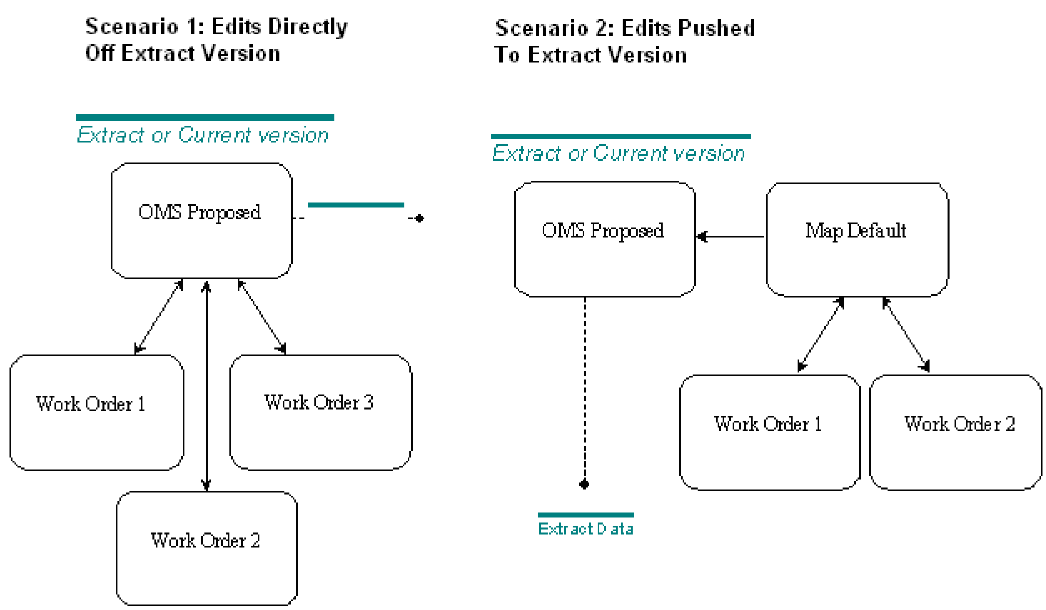
Versioned Database

For versioned databases, it is assumed there is a current or extract version (see diagram below). This could either be the SDE.DEFAULT version, or another version, so long as the following three criteria are met:

- The current or extract version contains the most recent data and serves as the source for the extract (i.e. proposed OMS data, ready for a model build).
- The current or extract version is of type public and not protected, so that permissions will not become an issue for extract users.
- The current or extract version is persistent (i.e. not a temporary version).

Other versions may exist in the database, such as checked-out versions for “in-progress” work orders, or a parent version that serves as a source for all the data (i.e. in a typical enterprise GIS database solution).

Two simplified scenarios for data maintenance are presented below. The first represents a system by which data changes are posted or made directly on the extract version, while the second represents a system where data changes are made on another version and that data is then transferred to the Oracle Utilities Network Management System proposed version for subsequent extract.



Supporting Oracle Utilities Network Management System Data Requirements

This section discusses how ArcGIS is used to provide mandatory and optional data for Oracle Utilities Network Management System.

Data for All Features

The following table provides descriptions for data that is essential for all features.

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Object type	Object type and/or attribute	Table name is the object type. The object type can also be derived from GIS object type and attribute values. See "Supporting Oracle Utilities Network Management System Symbology".
Object geometry (point, line, text)	Object shape	Adapter produces diagram object of type point, line, or text.
Points: (x,y) location angle, height	Object shape (junction)	Adapter produces diagram object characteristics of: location, angle, scale. Note that a switch is a junction feature.
Lines: List of coordinates	Object shape (edge)	Adapter produces diagram object characteristics of: location, angle, scale
Text: (x,y) location angle or endpoint string height width justification	Object shape	Adapter produces diagram object characteristics of: location, angle, string, size (height/text combination), and justification.

The following table provides descriptions for data that is optional for all features.

Oracle Utilities Network Management System Requirement	GIS Source	Comment
Object unique ID	FacilityID (ArcFM)	Alternatives are: ArcGIS ObjectID (unique within the class)
Diagram declutter ranking value	attribute	
(x,y) offset for graphical attributes		

Oracle Utilities Network Management System Requirement	GIS Source	Comment
Diagram groupings for multi-object hide/display		
Symbology class		
Scale factor	Refer to Supporting Oracle Symbology	
Object location ID	E.g., urban or rural attribute prescribes a certain scale factor	
Aliases (common names)	Based on Partition Tile	
Other operationally useful attributes	attribute	

Additional Data for Electrical Objects

This section discusses how ArcGIS is used to provide additional attributes for Oracle Utilities Network Management System electrical objects.

Additional Essential Attributes for Electrical Objects

The following table provides descriptions for additional essential attributes for electrical objects.

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Connectivity information	Node topology	
Normal status of device	NormalPosition[A,B,C] (ArcFM)	

Additional Optional Attributes for Electrical Objects

The following table provides descriptions for additional optional attributes for electrical objects.

Oracle Utilities Network Management System Requirement	GIS Source	Comment
Phases for the device (PHASES)	PhaseDesignation (ArcFM)	
Network protection direction	Attributes or relationships	
Operations control authority zone (NCG)	attribute	See section on NCG below. Required for all electrical objects including modeling nodes.

Oracle Utilities Network Management System Requirement	GIS Source	Comment
Voltage level (VOLTS)	NominalVoltage (ArcFM) or OperatingVoltage (ArcFM)	
Feeder (or Circuit) Name	Feeder/Circuit ID Attribute	Essential for feeder-based extraction

Supporting Network Component Group (NCG)

The Network Component Group (NCG) is an attribute of an electrical component in the Oracle Utilities Network Management System model. It supports control zones and reporting of events. The NCG value generally relates to a political structure.

The NCG value is generated by relating a GIS attribute (or combination of attributes) to a control zone hierarchy. The control zone hierarchy typically relates to political/administration structures such as:

- Company (highest level)
- Division
- District
- Office
- Substation
- Feeder segments (lowest level)

In the political structure in the example above, feeder segments are shown. This is a political structure imposed on the electrical structure of a feeder. NCG values correspond to a single hierarchy. Multiple inheritance is not supported so it is not possible to have an NCG value correspond to an electrical hierarchy (e.g., feeder, station) and a political structure. Where a feeder crosses political boundaries, it must be considered as a feeder segment belonging to one office, and other feeder segments belonging to other offices.

To provide enough information to generate the NCG, the GIS must have attributes to relate to the feeder segment. For example, a feeder name and office name (populated by a polygon overlay of the office polygons on the electrical components) can be combined into a unique NCG value.

Additional Data for Storm Management Support

These attributes are required to support Storm Management analysis. They apply to any object that can have an outage on it (e.g., a distribution transformer or breaker):

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Physical location	Varies according to class of object. Examples are: Three-phase devices are mapped to “backbone”. Single-phase devices are mapped to “lateral”. Distribution transformers are mapped to “transformer”.	The possible non-exclusive choices are: no_properties backbone express feeder interconnected lateral overhead secondary spot_network service sub transformer transmission underground Presently, only these choices are applicable: backbone lateral service transformer

Additional Data for Supporting Power Flow Analysis

These attributes are required to support Power Flow analysis.

Oracle Utilities Network Management System Requirement	GIS or Existing Power Flow Analysis Source	Comments
(To be determined)		

Oracle Utilities Network Management System Electrical Model Items

The Oracle Utilities Network Management System data model requires the following modeling items:

Oracle Utilities Network Management System Requirement	ArcGIS Source	Comments
Injection source point-Source Node (SRC)	CircuitSource	
Customer connection point-Supply Node (SND)	DistributionTransformer or equivalent	
Feeder start and boundary points –Feeder ID Node/ Feeder Boundary Node (FID/FBD)	CircuitSource or CircuitBreaker	GIS source objects imply the Oracle Utilities Network Management System nodes. The FBD and FID are generated by the preprocessor.
Mesh network and boundary points-Mesh Network ID Node (MID/ FBD)	PowerTransformer	GIS source objects imply the Oracle Utilities Network Management System nodes. The FBD and MID are generated by the preprocessor.
Inter-map electrical common points-Boundary Node (BND)	Not applicable in an ArcGIS geodatabase.	Generated automatically where tile boundary splits a conductor.
Groundable nodes (GND)	Ground	

Modeling Stations and Substations

There are three options for modeling stations and substations. The recommended option is shown first.

- **Option 1 - Draw Miniature Station**
- **Option 2 - Draw Stations in Separate Drawing**
- **Option 3 - Do Not Bother Modeling the Station**

Option 1 - Draw Miniature Station

In the GIS draw the substation within the footprint of the substation symbol (a rectangle). Draw it with complete and correct connectivity showing breakers and station transformers.

This option is easiest for the GIS to handle because it keeps the substation in the same coordinate space as the rest of the geodatabase features.

Oracle Utilities Network Management System can be configured to only show the tiny details when zoomed in.

Option 2 - Draw Stations in Separate Drawing

This requires common devices between the two maps, probably a breaker. Since the GIS does not support having the same object in two places, two separate breakers with a common attribute (plant identifier) are needed to link the two drawings.

Oracle Utilities Network Management System will generate a BND node to link the breakers.

Option 3 - Do Not Bother Modeling the Station

This is the easiest to model but has the major disadvantage of not being able to model cases where a station bus is out. It also makes it tricky to relate the implied breakers to the SCADA system's breakers so outage status due to a breaker trip can be properly captured.

A "getaway" point, with attributes of a feeder name, must be defined in the GIS for each feeder leaving a station.

Oracle Utilities Network Management System can be configured to convert the getaway point into a SRC, breaker, and other modeling nodes.

Modeling Spatially Dense Objects

Objects tend to be spatially dense in substations (covered above) and in vaults and manholes. A similar approach to the substations, using miniature switches (scaled symbols), can be used or liberties taken in the GIS to layout the vault switches in a schematic near the vault's spatial location.

Modeling Complex Objects

Oracle Utilities Network Management System models objects as nodes or edges in the same way as ArcGIS has junction and edge features. Complex objects are those with more than two connections such as 2-way and 4-way switches.

The simplest approach is to model in the GIS the complex objects by their component objects. For example, the 2-way switch would be modeled as three switches with interconnecting bus work surrounded by a container. The name of the component switches would be derived from the name of the 2-way switch (e.g., "S123-1" is one of the internal switches in 2-way switch "S123").

The other approach to model the complex objects is to use the Oracle Utilities Network Management System "aggregate" functionality.

Modeling Implied Objects

Oracle Utilities Network Management System can generate implicit objects. For example, a transformer object has elbows in the field, but often they are not modeled in the ArcGIS model. The Oracle Utilities Network Management System Model Build preprocess can create implicit objects such as elbows, which are needed for operating the electrical system.

In Oracle Utilities Network Management System, these implied objects can be given unobtrusive symbols so that the Oracle Utilities Network Management System appearance of the model is similar to the GIS appearance.

Mapping GIS Classes to Oracle Utilities Network Management System Classes

Options

GIS classes can be mapped to Oracle Utilities Network Management System classes in a number of ways:

- GIS class becomes one Oracle Utilities Network Management System class.
- GIS class is ignored. For example, the GIS SupportStructure class (pole) can be ignored.
- GIS class becomes an abstract Oracle Utilities Network Management System class, and the GIS subtypes become Oracle Utilities Network Management System classes. For example, the GIS DistributionTransformer has subtypes of SinglePhaseOverhead and ThreePhaseUnderground. The corresponding Oracle Utilities Network Management System classes can be SinglePhaseOverhead and ThreePhaseUnderground.

- GIS classes become one Oracle Utilities Network Management System class. For example, the GIS classes of SinglePhaseOverheadVoltageRegulator and AutoTransformer can map to Oracle Utilities Network Management System VoltageRegulator.

GIS annotation classes are also mapped to Oracle Utilities Network Management System classes in a similar way. GIS annotation is either simple text annotation or feature-linked annotation.

Feature-linked annotation is defined by an annotation class and the class of the feature it uses for the data source. It can only be defined for a concrete GIS class (it cannot be defined for a specific subtype of a GIS class). For example, annotation for the GIS DistributionTransformer can be mapped to an Oracle Utilities Network Management System class DistributionTransformerAnnotation. A Oracle Utilities Network Management System class based on the GIS DistributionTransformer's subtype, e.g., SinglePhaseOverhead, would have its annotation in the Oracle Utilities Network Management System class DistributionTransformerAnnotation. In other words, there would be no SinglePhaseOverheadAnnotation class. To support this requires substantial additional customization services of the Adapter.

Techniques for Retrieving the Object Model Schema

There are several ways for retrieving the GIS object model schema:

- Visually inspecting it using ArcCatalog
- Using RDBMS tools
- Using Microsoft Access tools.

The schema can be retrieved from the RDBMS by exporting the database and examining the schema found in the export.

During the process of defining the GIS object model, a geodatabase is built in a Microsoft Access database. Access has a function for analyzing the schema and reporting it:

```
Tools->Analyze->Documenter
```

The tables holding features are selected and the options for the analysis are:

- table's properties
- field names, data types and sizes
- nothing for indexes.

The resulting report can be exported into a spreadsheet where it can be cleaned up, sorted, and filters set for the columns. Additional columns of OMS and NOTES are added.

For example:

Class	Attribute Name	Type	Size	OMS?	Notes
CapacitorBank	(Class)			Y	
CapacitorBank	OBJECTID	Long Integer	4	N	
CapacitorBank	Shape	OLE Object	-	N	
CapacitorBank	HighSideConfiguration	Text	1	N	
CapacitorBank	LocationType	Text	3	N	
CapacitorBank	PhaseDesignation	Long Integer	4	Y	

Class	Attribute Name	Type	Size	OMS?	Notes
CapacitorBank	RatedVoltage	Long Integer	4	N	
CapacitorBank	OperatingVoltage	Long Integer	4	Y	
CapacitorBank	LabelText	Text	50	Y	
CapacitorBank	TotalKVAR	Long Integer	4	Y	
CapacitorBank	SubtypeCD	Long Integer	4	Y	
CapacitorBank	FacilityID	Text	20	Y	
CapacitorBank	InstallationDate	Date/ Time	8	N	
CapacitorBank	SymbolRotation	Double	8	N	
CapacitorBank	WorkOrderID	Text	20	N	
CapacitorBank	FeederNumber	Text	20	N	
CapacitorBank	CustomerOwned	Text	1	N	
CapacitorBank	DeviceLocation	Text	70	Y	
CapacitorBank	CreationUser	Text	10	N	
CapacitorBank	DateCreated	Date/ Time	8	N	
CapacitorBank	DateModified	Date/ Time	8	N	
CapacitorBank	LastUser	Text	10	N	
CapacitorBank	AncillaryRole	Integer	2	N	
CapacitorBank	Enabled	Integer	2	N	
CapacitorBank	SubtypeCD=FixedBank			Y	OMS class FixedCapacitorBank
CapacitorBank	SubtypeCD=SwitchedBank			Y	OMS class SwitchedCapacitorBank

Adapter Functionality

The adapter program extracts objects and features from the ArcGIS database and produces extract files in Oracle Utilities Network Management System Model Preprocessor (".mp") format. In ArcGIS terminology, the program makes use of a data converter engine. The Adapter is called "AItoCentricity."

The Adapter makes use of ESRI's ArcObjects via a Visual Basic shell (COM program). Its extraction behavior is configured by default values set in an accompanying Microsoft Access database, which has the same name as the executable (i.e. AItoCentricity.mdb) and resides in the same subdirectory. This required database is the "application reference" database.

The Adapter can be installed and run on any PC with Windows 2000/Windows XP as the supported platforms. Note this application does not run on the SDE server, however it does use SDE resources through ArcObjects if the data is SDE-based.

The Adapter can be run against either a personal geodatabase or an SDE database. Since database versioning is not supported in a personal geodatabase, the Adapter's change detection mechanism will only work on SDE data that has also been registered as versioned.

There are two methods of invoking the Adapter:

- Non-interactively by a command line with parameters.
- Interactively with a GUI.

The program can be launched in non-GUI mode from an OS command prompt with command line parameters. The user can either use the defaults as set in the application reference database or override them with passed parameters. As such, the program can be scheduled for "silent" launch as a Windows scheduled task. Note because this type of launch is "silent" and does not make use of a GUI, feedback is usually not provided to the workstation screen unless an error occurs.

The program can be launched from the workstation desktop or pull down menu. Using the GUI, the user can interactively control what data is extracted, what extract method is to be used, the type of data to extract, and the name of the output directory. The user is also provided with a status area, which includes progress bars and messages for monitoring the progress of the overall workorder and the specific partition currently being extracted.

Adapter Capabilities

The Adapter extracts information from the GIS in groups of features known as "partitions"; all of the features in a partition, along with an overall partition object that defines an enclosing rectangle for the contained features, are placed in one extract (.mp) file. The process of grouping features into partitions can be done in one of two ways:

- Geographic, tile-based partitions can be defined using a grid feature class that exists within the GIS. With tile-based partitions, all of the features contained within or overlapping one particular "cell" or region as defined by this grid feature will be grouped into the same partition by the adapter and will be extracted into the same extract file.
- Non-spatial, attribute-based partitions – e.g., feeder/circuit-based partitions -- can be defined by supplying the name of an attribute (or two attributes in the case of feeders when a device might have two feeders because it acts as tie-point between feeders e.g., "FEEDERID1" and "FEEDERID2"); the distinct values of this attribute (e.g., all the distinct values for feeder ID) will define the partitions: all the objects sharing a particular value of this attribute (e.g., all the features belonging to a particular feeder/circuit) will be grouped into the same partition by the adapter and will be extracted into the same extract file. (Optionally, if the "SpatialWithNonSpatial" option in the "Defaults" table is checked, features lacking this characteristic attribute but falling within the same bounding rectangle will also be extracted. Typically, this option would be enabled if you want to extract non-electrical features like support structures and annotation that are not marked as belonging to a particular feeder but

which you would like to extract into the same partition as the adjacent electrical features that do belong to a feeder.)

Configuring which of these two methods of defining a partition should be used is done primarily through the "PartitionReference" table in the application reference database as explained in detail below in the "Adapter Configuration" section of this document.

Optionally, the Adapter can be used to detect changes to a geodatabase, thereby automatically flagging partitions for extraction.

Automated change detection by the Adapter is based on ArcGIS version functionality. Accordingly, the Adapter manages the internal versions it makes use of to support its change detection. However, for those clients that prefer not to implement ESRI SDE versions in their database, the Adapter can be configured to read an externally generated list of changed partitions and bypass creating/using SDE versions. This functionality permits those clients to benefit from the change detection functionality (e.g., for incremental model builds), however it clearly places the onus of change detection on the data maintenance team.

If change detection by the Adapter is chosen, then a versioned database must be used. As a result, the Adapter will create a temporary extract "freeze" version for the extraction process. If the extract completes successfully, this temporary version is then saved as a valid data freeze for future use. Then, on the next launch of the Adapter, it will attempt to intelligently detect the last successful freeze version (based on creation and modification dates). If located, it will highlight the last freeze version for use with the change detect tool. The change detect tool is an optional automated mechanism for queuing partitions that contain objects and features that have been either changed, added, or deleted since the last extract.

On extraction, the Adapter retrieves:

- Partition details.
- Objects. For each object:
    - Object class and identifier.
    - Connectivity.
    - Geometric information.
    - It retrieves rows and attributes for the object as well as from related or connected features and tables. For example, the related instance(s) of TransformerUnit may be extracted when the instance of Transformer (a transformer bank) is extracted.
    - Feature-linked annotation. This is a special case of a related table.
- Labels and Annotation.

The Adapter produces output in a work order folder containing extract files in Oracle Utilities Network Management System Model Preprocessor (".mp") format and a log file of informational, warning, and error messages.

Partition Details

The Adapter generates a "partition section" object as the first object in each MP file. This partition object is based on the geometry of the tile or the minimum-bounding rectangle for the objects identified as belonging to a particular feeder (for feeder-based extracts). In the case of tile-based extraction, it uses the specified grid feature class as the class name for this partition and an attribute (e.g., "TileID") for the partition's name. For feeder based extraction, this has an object type of "NONSPATIAL" and an object ID equal to the distinct attribute value – i.e. the FEEDERID for this partition.

Object Extraction

The Adapter generates object records for each object. The object records add the object, its topology, Diagram records, and Attribute records.

Object Identifier

The Adapter uses the feature class name and feature ID to identify an object. For example, “Switch 24.” Note the feature ID is unique within the feature class.

Connectivity

The Adapter determines the connected (or networked) objects by analyzing the ArcGIS’ Geometric network. The connectivity (topology) is established using the EID attribute, a unique index number in the Geometric network. The Adapter translates the EID into a port value that contains the following:

- Feature Class Name
- Object ID
- Sub ID

For example, 6.324.1 could be PriOhElectricLineSegment OID #324, Sub ID #1.

The Oracle Utilities Network Management System AttTopoBuild program uses the port numbers to generate topology.

To support identification of this feature, an attribute object FEATURE_CLASS_ID is produced. This, in combination with the objectid.subid can be used as an identifier equivalent to the port identifier of connected objects.

Geometric Information

The Adapter determines the geometry of the object and uses it for the coordinates of the object as well as for any Oracle Utilities Network Management System diagram objects generated from the object: scale, height, angle, etc. The geometric diagram line should specify the feature class name.

Related Information

Although “core” attributes will always be output for an object, the Adapter will only traverse to related objects if the class relationship is listed in the “RequiredRelationships” table of the application reference database. This is to enhance performance and minimize file sizes. For convenience, attribute records (identified by feature class name or object class name) are output in the following order:

- Attributes
- Attributes from related tables

Coded Value Look Up

The Adapter can be configured to either provide the raw code for an attribute with a coded-value domain from the database or to perform a “look up” for the coded value. This option can be set in the “Defaults” table of the application reference database.

If the look up is not used, it requires the Model Build process to perform the lookup. Two options are:

- Model Build preprocessor contains a copy of the coded value domain tables and looks it up.
- Copy the lookup table to the OMS and use a database view to bring them together for presentation. This view slows down the Oracle Utilities Network Management System attribute display so it is not a recommended approach.

The obvious disadvantage of both these methods is maintaining and synchronizing lookup tables in both the GIS and OMS.

Decomposing Complex Edge Features

Oracle Utilities Network Management System requires simple features, so complex edge features in ArcGIS data are decomposed into their components: simple edge features. Each component gets an Object record with the attributes of the complex edge. Since we are breaking a complex edge into multiple simple edges it will require recalculation of “MeasuredLength” attribute for each of the simple edge. The “MeasuredLength” attribute could be derived from the complex edge feature’s “MeasuredLength” attribute using the ratio of the “Shape_Length” of the simple edge feature to the “Shape_Length” of the complex edge feature. The Shape_Length is an ArcGIS geographic attribute that provides a length of an edge.

Instead of calculating “MeasuredLength” attribute using the method described above, the Adapter uses the simple edge’s “Shape.Len” attribute value to populate the measured length attribute.

By default, feature-linked annotation for complex objects is always associated with the first component of the set of complex edge features when output in the extract file. However, the geometry for the annotation feature is maintained so the annotation feature’s spatial placement in the OMS should always be consistent with the GIS’.

Class Names Containing Space Characters

It is possible for GIS class names to contain spaces in them (e.g., “Overhead Line”), particularly in a personal geodatabase. However, the Oracle Utilities Network Management System model build tools do not recognize or accept class names with spaces, so a conversion must be made.

The accepted convention is that all spaces be substituted with underscores (e.g., “_”). Therefore, if source data class names have spaces in them, the final output model preprocessor (.mp) files will have underscores substituting the spaces. An example would be a class named “Overhead Line”. The ADD and ATTRIBUTE statements in the extract output file would show the name “Overhead_Line” instead.

Labels and Annotation

Extracting a label as a separate feature is possible, but it is not recommended because Feature and Label object data does not have any explicit relationship. Without the explicit relationship, Oracle Utilities Network Management System Preprocessor and Model Build process cannot associate the label text with the feature class in the Oracle Utilities Network Management System viewer leading to inefficient behavior of the hide/display operation. For example if a Switch and Switch text are not associated, then turning off display of Switch using the hide/display tool, will not automatically hide the associated text as well. Feature-linked annotation provides similar functionality and should be used instead since it maintains the relationship between a feature and its annotation.

Feature-linked annotation is treated as an attribute of the related object. The text and geometry associated with the annotation is extracted and expressed as a text diagram object. The extraction executes the governing expression used in defining the annotation.

The advantage of feature-linked annotation is that annotation can be grouped with the feature in Oracle Utilities Network Management System for decluttering purposes.

Change Detection

Change detection is a feature that automates the queuing of partitions for an extract. Its purpose is to detect and queue for extract those partitions that contain data (objects) that have been modified, added, or deleted since the last successful extract and build. As such, the user can either use the Adapter’s built-in change detection functionality, or read the results from an externally generated change detection source. Each of these options is discussed in the sections that follow.

Change Detection by the Adapter

Change detection performed by the Adapter will only work on ESRI SDE versioned data.

On the first run of the Adapter, a change detect is not possible because version maintaining for the previous data state will not yet exist (i.e. no freeze yet exists). However, once the first full extract is completed, a freeze version will exist for a change detect. Then, all subsequent successful extractions will create new freezes that represent the state of the data at the last extract for use in the next change detection.

On any subsequent launch of the Adapter, the most recent freeze version will be automatically selected in the change detect drop down list. This is the recommended and normal approach to using Change Detection.

The only exception is if a specific change detect version is listed in the “ChangeDetectVersion” field of the “Defaults” table in the application reference database. Normally, this field would remain blank so that the proper freeze version can be selected. However, this ability has been placed in the Adapter for those clients that choose to reuse a version for maintaining state on the last successful extract.

See **Freeze Versions Created and Used by the Adapter** on page 4-38 for more details.

Change Detect in a Multi-Schema ArcSDE Environment:

The Adapter uses the ArcObject version reconciliation method to identify the changed classes between two versions. The ESRI tools do not provide a way to specify a schema on which change detect should be performed, so reconciliation method identifies all the changed classes in various schemas. For example, if Electric and Gas schemas are created on the same ArcSDE database server than Adapter change detects will identify changed classes in both Electric and Gas schemas.

The Adapter’s change detect method is enhanced to identify only changed classes (and thus changed partitions) in a user-specified schema. The user specifies a schema indirectly by providing the Dataset name, which belongs to the schema. The Adapter detects changes only in the schema of dataset listed in the DatasetReference table.

The Adapter’s change detect functionality pre-loads the class name for each dataset and then cross-reference the changed class names as they are encountered, thus avoiding feature class/ schemas which are not included in the datasets listed in the DataSetReference table.

Change Detection from an Externally Generated Source

This option assumes that change detection has been performed prior to the Adapter being launched, and that the results are available for the Adapter to read. Usually, this approach means that change detection is performed either as part of data maintenance, or as a batch job prior to extraction.

Out of the scope and control of the Adapter, the results of external change detection are placed in a table (with specific schema) in the GIS database. The Adapter is then configured to “read” the results from this table, and provide the GUI with the “partition sets” as defined in the table. In the GUI, the user can then choose one of these defined sets or queues, and apply this filter set to the existing list of partitions to queue partitions for an extract.

Note that maintenance of the externally generated change detect or partition queue table is out of the scope of the Adapter, and alternate tools must be in place to manage the data (i.e. the partition queue table).

Installing ArcGIS to Oracle Utilities Network Management System Adapter

There are four components to installing the Adapter: system configuration, application installation, the application reference database file, and optional testing.

- **System Configuration**
- **Installing the Application**
- **Application Reference Database File**
- **Testing the Installation**

System Configuration

Environment Variable

If ArcGIS is not on the Adapter workstation, then add the following environment variable:

```
ESRI_LICENSE_FILE = [<port>]@<server>
```

Where `<port>` is optional, and `<server>` is the name of the ArcGIS license server in your installation (e.g.- ESRI_LICENSE_FILE=@boron).

```
Windows NT: Start - Settings - Control Panel - System - Environment

Windows 2000: Start - Settings - Control Panel - System - Advanced
 - Environment Variables
```

Hosts File

If ArcGIS is not on the Adapter workstation, then add the port and host information for all required ESRI clients and servers:

```
192.37.0.101boron# ArcGIS development PC
192.37.0.105gis_server1# ArcSDE server

Windows NT: C:\winnt\system32\drivers\etc\hosts
Windows 2000: C:\windows\system32\drivers\etc\hosts
```

Services File

If ArcGIS is not on the Adapter workstation, then add the ESRI/SDE database services and port information:

```
esri_sde5151/tcp# ESRI SDE port for SDE default database
Windows NT: C:\winnt\system32\drivers\etc\services
Windows 2000: C:\windows\system32\drivers\etc\services
```

Extract Output Base Directory

Ensure the path specified in the “ExtractDirectory” field of the “Defaults” table of the application reference database exists and is accessible from your client PC. This directory is required, so create it if it does not exist.

Note this may be a shared or mounted network drive (e.g., “\\data1\extracts”) so long as the workstation has access through the operating system. Ensure access to this drive will not be interrupted during extracts (e.g., archiving or network maintenance “down times”).

Installing the Application

Since the Adapter makes use of ESRI's ArcObjects, it assumes there is a valid ESRI ArcGIS license installed either locally on the PC or on another network accessible computer. Otherwise, the Adapter will not work.

If a previous version of the Adapter exists on the workstation, uninstall the previous version first. However, be sure to keep a backup copy of your application reference database file for use with the new software. Notice of schema changes for this database should be found in the readme.txt file in the installation directory.

If this is the first installation, then an empty application reference database file can be obtained from the installation package. Refer to the section below labeled "Application Reference Database File" for details on this required file.

Be certain to install the software while logged on to the workstation as an administrator.

Obtaining the Software

The adapter setup files are included in the Oracle Utilities Network Management Optional Windows Applications download zip file that can be obtained from the Oracle e*Delivery site. After downloading the zip file, find the appropriate Adapter zip file within the NMS_GIS_Adapters directory and unzip to a location on your Windows server.

Automated Installation

Run (double click) the setup.exe file found in the downloaded software at:

```
AItoCentricity_1_9_0\setup.exe
```

Follow the instructions and prompts as required. Try to use the defaults unless they conflict with your systems configuration (e.g., installation directory).

Note the Adapter requires the Microsoft Data Access Components (MDAC version 2.7) product be installed on the workstation, since it contains drivers required for querying the application reference database file (in Access 2000 format). If these drivers are not found on your workstation, the package will automatically install this Microsoft product for you. On completion you may need to reboot the workstation and then re-launch the installation package to continue installing the Oracle Adapter software.

Application Reference Database File

The Adapter requires an application reference database file ("AItoCentricity.mdb"). This file is used to store the custom parameters and default configurations required for the Adapter to operate at your specific location. A primary copy of this file should always be placed in the same directory as the Adapter software.

Note that copies of this file configured with different default parameters may also be stored elsewhere on the workstation. The most likely purpose for these additional reference database files would be to provide different defaults for non-GUI extracts.

An empty application reference database file can be found in the installation source at:

```
AItoCentricity\support\AItoCentricity.mdb
```

Be sure to populate this file according to your specific requirements. See the Adapter Configuration section below for details.

Testing the Installation

The Adapter can be tested on an Access database or an SDE database. This section describes a quick test that can be conducted on an Access geodatabase.

Before running the Adapter on an Access database, make sure the dataset contains network information. Without this, the extract will have no topology information.

There is an ArcMap wizard for generating a geometric network. See *Creating Geometric Networks* in ESRI's Building a Geodatabase document for details.

The GIS data model should specify what features belong to the electrical network. This specification can be used to answer the wizard's questions.

Typically, the answers include:

- Putting all electrical features into the network.
- Using complex edges for conductors. This is the preferred method from a GIS perspective for minimizing the number of features.

Configuring the Adapter

The Adapter is configured using tables in a Microsoft Access database, commonly referred to as the "application reference" database. The specific configuration tables, their contents, several examples, and content requirements are provided in the sections that follow.

Reference Database Tables

Defaults Table

The Defaults table is used to indicate the default settings to use for the Adapter. These settings should reflect the most common parameters used for an extract:

Parameter	Value	Type
ExtractDirectory	Base/Root Directory for Extracted Data Output	Text(50)
PartitionType	Name of Default PartitionType/ExtractType combination to use	Text(50)
ExtractType	Type of Extract to Perform (sourced from field "ExtractType" from "DatasetPartitionTypeExtractType_Relation" table)	Text(50)
ExtractVersionName	Name of Version to Extract	Text(50)
ChangeDetectVersionName	Name of Version to Perform Change Detect Against (optional - populate this only if change detection is performed by the Adapter)	Text(50)
UseCodedValues	Extract Attributes as Coded Values Instead of Raw Data (True/Check=Supply Codes, False/Uncheck=Supply Raw Data)	Boolean
UseSDEVersioning	Specify Whether to use SDE Versions or Not (True/Check=Use/Create, False/Uncheck=Don't Use/Don't Create)	Boolean

Parameter	Value	Type
NumDaysToKeepFreeze	Number of Days to Keep a Database Freeze Before Deleting (0 means forever)	Integer
RelatedFeatureClassLevels	Number of Levels to Traverse when Handling Related Feature Classes	Integer
RelatedObjectClassLevels	Number of Levels to Traverse when Handling Related Object Classes	Integer
NonFacilityExtractFileSuffix	Suffix to Place on mp Extract Files for Non-Facility Extracts	Text(20)
ExtractFilePrefix	Prefix to Place on mp Extract Files	Text(20)
LatLongForPoints	Specify whether to Provide Lat/Long Attributes For Point Features	Boolean
SpatialAttributesForPoints	Specify whether to Provide Spatial Attributes For Point Features (as defined in the optional “SpatialAttributeQueries” table, and as coded in the functionality for obtaining the “closest service address” results for a point.)	Boolean
UsePartitionQueue	Specify whether to use PartitionQueue table partitionset to populate the GUI Partition listbox or not.	Boolean
ExtractPolyline	Specify whether to extract poly line coordinates using From and To pair of coordinates or just one pair of coordinate. Default is False, i.e extract only one pair of coordinates. If GIS has poly line with gap, it is recommended to extract both pair of coordinates	Boolean
ContinueExtract	Specify whether to continue extraction even when one of the partitions fails to extract. This option is helpful for a batch process when instead of stopping whenever extraction fails on a specific partition extract moves on to next available partition in the queue.	Boolean
NormalizePort	If true, then normalize port id; if false, then port number is in featurenumber.objectid format.	Boolean
SpatialWithNonSpatial	If true, then do a spatial extract for features within MBR after doing non-spatial extract.	Boolean
GetAnnoOnRelated	If True, then extract all related Annotation Features	Boolean
GetRelatedAnnoSpatially	If True, then also check for Related Annotations, to be extracted spatially	Boolean
VersionNamePrefix	Specify Freeze Version Prefix, for Electrical & Non-Electrical Change Detect process	Text(50)

Parameter	Value	Type
GisDataOwner	When specified, allows extraction from different GIS user names. Applicable mainly during extraction of related features	Text(50)
ExtractNullFeeder	In non-spatial extracts, extract objects with NULL values for both feeder attributes into aq feeder named NULL_NULL	Boolean
SkipEmptyPartitions	If an extraction .mp file does not contain any objects, by default, the empty .mp file will be generated with the partition definition only. With this option checked, the empty partition will not be written out.	Boolean
LeaveSubtypeFieldNames	If UseCodedValues is true and this is true, the SYBTYPECD field is not renamed to SUBTYPE.	Boolean
UseRelationshipSets	If true, extract using relationship sets instead of getting one relationship at a time.	Boolean

Note: If the “SpatialAttributesForPoints” Boolean is checked (yes or true), ensure the optional SpatialAttributeQueries” and “ClosestServiceAddress” tables exist and are properly populated.

Parameter Values Examples

Parameter	Value
ExtractDirectory	“c:\SPL”
PartitionType	“Electrical”
ExtractType	“Facility”
ExtractVersionName	“OMS Proposed”
ChangeDetectVersionName	“” (blank means use the last freeze version)
UseCodedValues	✓ (a check means to “use” or “extract” coded values)
UseSDEVersioning	✓ (a check means to “use” or “create” SDE versions)
NumDaysToKeepFreeze	14 (keep freeze versions for 14 days)
RelatedFeatureClassLevels	2
RelatedObjectClassLevels	1
NonFacilityExtractFileSuffix	“_lb”
LatLongForPoints	✓ (a check means to “use” or “extract” lat/long coordinates as extra attributes on point features)

Parameter	Value
SpatialAttributesForPoints	✓ (a check means to perform the spatial attribute queries, as defined in the optional SpatialAttributes table. If checked off here, ensure the “SpatialAttributeQueries” and “ClosestServiceAddress” tables exist and have been populated in the application reference database.)
UsePartitionQueue	✓ (a check means to “use” PartitionQueue table to read list of partitions to extract instead of defined Partition class in the geodatabase)
ExtractPolyline	✓ (a check means to extract From and To pair of coordinates for a polyline.)
ContinueExtract	✓ (a check means continue to extract even when one of the partitions fails to extract)
NormalizePort	✓ (a check means normalize the port)
SpatialWithNonSpatial	✓ (a check means do a spatial extract for features within MBR after doing non-spatial extract.)
GetAnnoOnRelated	✓ (a check means, extract all related Annotation features)
GetRelatedAnnoSpatially	✓ (a check means, spatially extract related Annotation features, if stored spatially
VersionNamePrefix	“_elec” for Electrical freeze version or “_nonelec” for Non-electrical freeze version
GisDataOwner	Specify GIS user name for the extractor to access, other the default user name. Used mainly for related features.
ExtractNullFeeder	✓ (a check means, extract NULL-NULL feeder)

DataSet Reference Table

The DataSetReference table is used to indicate the datasets (data groupings) found within the database and their appropriate configurations:

Parameter	Value	Type
DataSet	Name of Dataset to Reference From the Workspace (e.g., Electric, Landbase)	Text(40)
DatabaseName	Database File Name (geodatabase only)	Text(100)
Instance	Database Instance Name	Text(50)
UserID	Database userid for the Instance	Text(50)
Password	Database password for the userid	Text(50)
ConnectionType	Data Connection Type (ACCESS, SDE)	Text(10)
Server	Database Server Name	Text(30)
Version	Database Version to use for Extracts	Text(50)

For example, an SDE dataset of electric feeder facilities is defined:

Parameter	Value
DataSet	“Electric_Feeder”
DatabaseName	“” (this is only for a path for a personal geodatabase)
Instance	“esri_sde”
UserID	“gisDBuser”
Password	“gisDBuser_password”
ConnectionType	“SDE”
Server	“gis_server”
Version	“OMS Future”

Partition Reference Table

The PartitionReference table is used to indicate the partitions (spatial, geographic tiles or attribute-defined, e.g., feeder-based, groupings) to be used for extraction and their appropriate configurations:

Parameter	Value	Type
PartitionType	Unique Name of the Partition Type (e.g., Landbase, Electrical, Feeder, etc.)	Text(50)
ClassName	Name of feature class (or classes) used to define partition (e.g., tile or grid-feature class name for spatial extract vs. objects that define origin of feeder/circuit for feeder-based)	Text(50)
IDColumn	For spatial extract: Name(s) of Partition ID Column(s) (in partition class) separated by a comma. For non-spatial extract: Name(s) of characteristic attribute column(s) (such as FEEDERID) separated by commas	Text(30)
WhereClause	SQL where clause to limit the partition id selection	Text(80)
ExtractMethod	Partition Extraction Method (Spatial (i.e. tile) or NonSpatial (i.e. feeder))	Text(50)
DatabaseName	Database Name Partition is in (geodatabase only)	Text(100)
Instance	Database Instance Name Partition is In	Text(50)
UserID	Database userid for the Instance	Text(50)
Password	Database password for the userid	Text(50)
ConnectionType	Data Connection Type (ACCESS, SDE)	Text(10)
Server	Database Server Name	Text(30)
Version	Database Version Name the Partitions are In	Text(50)

For example, an SDE electric partition spatial feature class is defined:

Parameter	Value
PartitionType	"Electric_Partitions" or "FEEDER"
ClassName	"ElectricTiles" or "CircuitBreaker, SubTransBreaker"
IDColumn	"TileID,ObjectID" or "FEEDERID, FEEDERID2"
WhereClause	"TILID<1000 and OBJECTID>200"
ExtractMethod	"SPATIAL" or "NONSPATIAL"
DatabaseName	"" (this is only for a path for a personal geodatabase)
Instance	"esri_sde"
UserID	"gisDBuser"
Password	"gisDBuser_password"
ConnectionType	"SDE"
Server	"gis_server"
Version	"OMS Future"

Partition Queue Reference Table

The PartitionQueueReference table supplies the location details for the Partition Queue table (which is used to store and recall saved partition queues):

Parameter	Value	Type
DatabaseName	Database Name Partition Queue Table is in (geodatabase only)	Text(100)
TableName	Name of the Partition Queue Table (most likely "PartitionQueue")	Text(50)
UserID	Database userid for the Instance	Text(50)
Password	Database password for the userid	Text(50)
ConnectionType	Data Connection Type (ACCESS, SDE)	Text(10)
Server	Database Server Name configured using Oracle NetConfiguration utility.	Text(30)
ReadOnly	Specifies whether the PartitionQueue table can be written to, namely for the ChangeDetect Queue or Selection Queue.	Boolean

For example, for a local Access database partition queue table:

Parameter	Value
DatabaseName	“” (specify if in a database other than the application reference file)
TableName	“PartitionQueue”
Instance	“”
UserID	“”
Password	“”
ConnectionType	“ACCESS”
Server	“”
Version	“”
ReadOnly	“” (no check means the table can be written to)

For example, for an Oracle database partition queue table:

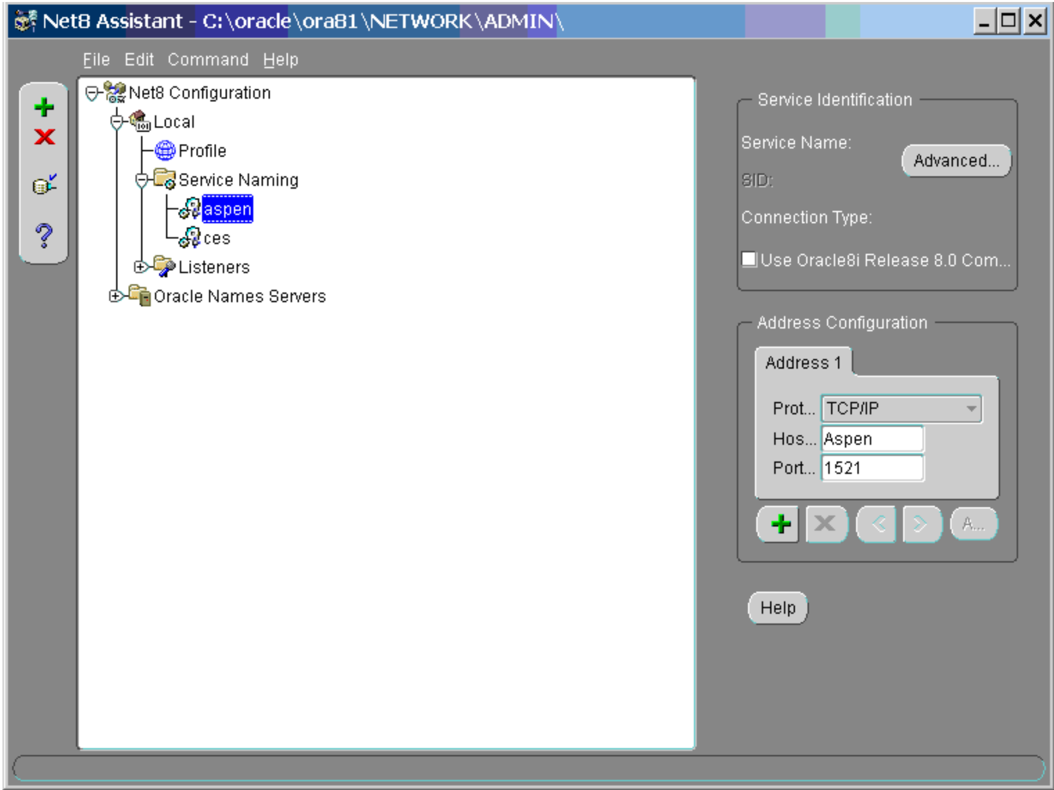
Parameter	Value
DatabaseName	“” (used only for an Access database or geodatabase only)
TableName	“PartitionQueue”
Instance	“”
UserID	“gisDBuser”
Password	“gisDBuser_password”
ConnectionType	“SDE”
Server	“gis_server" with "gis_server in the tns_names.ora configuration or "(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL =TCP)(HOST=nms-vm)(PORT=1521)))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAM E=nms11g)))”
Version	“”
ReadOnly	“” (no check means the table can be written to)

The contents of this table affect the GUI as follows:

- If no table name is specified, the “Partitions Selection Mode” panel, “Select Partitions by Filter” button, and “Save Selected Set to a Table” sections will be disabled, restricting the user to manual selection. The “Change Detection” panel will also be disabled since there would be no table available to save the results to.
- When “ReadOnly” is checked, the “Change Detection” panel and “Save Selected Set to a Table” sub-panel will be disabled, preventing the user from saving a Partition Set to the *PartitionQueue* table.

Oracle Client Configuration

If the connection type is “SDE”, i.e. the PartitionQueue table resides in Oracle database instead of AItoCentricity.mdb then the user must ensure that the appropriate Oracle client is installed on the machine where the Adapter resides. The user must configure the Server name using “Oracle Net8 Configuration Assistant”. The Net8 Assistant screen appears below:



DataSet Partition Type/Extract Type Relation Table

The DataSetPartitionTypeExtractType_Relation table is a configuration table used to indicate the relationship between “Partition Type”, “DataSet” and “Extract_Type”. This table is meant to clarify some of these ambiguous relations. The application will use these relations in order to populate its GUI.

The existing application allows the user to specify this relationship on each extract. Instead of using the run-time specification, the relationships are made explicit and treated as a configuration that defines the type of extraction to perform when combinations of these two aspects are chosen:

Parameter	Value	Type
DataSet	The proper ESRI dataset name. Values are restricted to enumerated “DataSet” values found in table 'DataSetReference'.	Text(40)
PartitionType	Unique name of the partition type. Values are restricted to enumerated “PartitionType” values in table 'PartitionReference'.	Text(50)
ExtractType	Type of Extract to perform for the dataset; “Facility” or “NonFacility.” Used to determine the suffix to place on .mp files.	Text(30)

For example, a dataset called "Electrical" and a partition type of "Electrical_Grid" should be a "Facility" extract:

Parameter	Value
DataSet	"Electrical"
PartitionType	"Electrical_Grid"
ExtractType	"Facility"

DataSet Exceptions Table

The DataSetExceptions table is used to indicate any exceptions when processing classes. The Adapter assumes a feature class will be *included* as part of an extract of the dataset it belongs to unless an exception record indicating it should be excluded from processing appears in this table. Similarly, the Adapter assumes that a feature class will be *excluded* as part of an extract of a dataset that it does not belong to unless an exception record indicating it should be *included* from processing appears in this table. In other words, do not list a feature class here unless you want it to be excluded from an extract of the dataset it belongs to, or included in an extract of a dataset it does not belong to.

Parameter	Value	Type
DataSet	DataSet to Provide Special Instructions For	Text(40)
ClassName	Name of Class to Include/Exclude from the DataSet (exceptions class)	Text(50)
Action	Action to Take With Specified Class (check=true=include,uncheck=false=exclude)	Boolean

For example, include a class called "WorkzoneTile" in the electrical dataset extraction:

Parameter	Value
DataSet	"Electrical"
ClassName	"WorkzoneTile"
Action	✓ (check means to include the class)

Required Relationships Table

The RequiredRelationships table is used to indicate which relationships between classes in the GIS database should be traversed in an extraction. This table should therefore indicate the class relationships required for the model build:

Parameter	Value	Type
OriginClass	Name of Origin Class (class that originates the desired relationship, or parent class to the direction of the relationship)	Text(50)
DestinationClass	Name of Destination Class (class that ends the relationship, or child class of the relationship)	Text(50)
RequiredForOMS	Whether Relationship is Required for OMS Extract (Yes/Check=Required For OMS, No/ Uncheck=Not Required For OMS)	Boolean

For example, a class relationship between “Fuse” and “FuseUnit” needs to be included in an extract for a successful model build:

Parameter	Value
OriginClass	“Fuse”
DestinationClass	“FuseUnit”
RequiredForOMS	✓ (check means to include the class relationship)

Partition Queue Table

The PartitionQueue table is used by the Adapter to store sets or queues of partitions for later recall and extraction. It can exist either in the application reference database or in the SDE geodatabase. If creating this table in your SDE database, please ensure the schema (field names and types) shown below are closely adhered to:

Parameter	Value	Type
PartitionSetName	Groups the Partitions Queue under a unique Partition Set Name	Text(50)
DateStamp	Date/Time stamp when the specified Partition was saved to this table	Date
DataSetName	The proper ESRI dataset name that corresponds to a specific Partition Name (1 DataSet per record)	Text(40)
PartitionName	Specific name of the partition in the Partition Queue	Text(50)
PartitionType	Unique name of the partition type that corresponds to a specific Partition Name	Text(50)
ExtractDate (provisional)	Date when the specific Partition Set record was extracted. Note: this field does not get updated if the “Read Only” field is checked in the “PartitionQueueReference” table.	Date

Note: The DATESTAMP field in the “PartitionQueue” table can be null in the ACCESS application reference database, but not in SDE/Oracle Utilities. For consistency, the best practice is to define the field as not allowing null but permitting zero-length strings (i.e. “”).

For example, the partition “774410-000000” was identified, queued, and saved as part of the set “SelectionSet_200208261412” on “August 28” for the dataset “Electric_Feeder”:

Parameter	Value
PartitionSetName	“SelectionSet_200208261412”
DateStamp	“26/08/2002 2:12:34 PM”
DataSetName	“Electric_Feeder”
PartitionName	“774410-000000”
PartitionType	“Electrical”
ExtractDate	“” (not extracted yet, or table is read only)

A specific “Partition Name” can have more than one related “DataSet Name”. Each DataSetName/PartitionName combination will correspond to one distinct record in the table. The same applies to the Partition Types.

SpatialAttributeQueries Table

The optional SpatialAttributeQueries table is used by the Adapter to store information on custom spatial queries. These spatial queries add spatial attributes to point features (e.g., a point in polygon analysis). Note that a record must appear in this table for every spatial query that a point feature class is to have applied to it. To use this table, make sure the “SpatialAttributesForPoints” Boolean is checked (yes or true) in the “Defaults table, otherwise this table will be ignored.

Parameter	Value	Type
AttributeName	Custom name of the attribute that should be appended to the point features in the extract file.	Text(50)
PointClassName	The point FeatureClass that will have this spatial query applied to it. Note that a spatial query can be repeated n times, once for each FeatureClass needing the query.	Text(50)
PolygonClassName	Name of the polygon feature class that the spatial query is being applied to.	Text(50)
IDColumn	Name of the column/field in the polygon feature class that should be “pulled” or applied as the spatial attribute for the point features.	Text(50)
WhereClause	Any characteristics for refining the spatial query (omit the WHERE statement from the field, it is assumed).	Text(100)

For example, using the polygon feature class called “FM_Counties” that has the unique field “Name”, perform a spatial query for adding the county name as the attribute “County” to the “Fuse” point feature class in the output .mp file:

Parameter	Value
AttributeName	“County”
PointClassName	“Fuse
PolygonClassName	“FM_Counties”
IDColumn	“Name”
WhereClause	“”

As an example, this spatial query would result in an attribute being added to a fuse feature in the extract (mp) file similar to:

```
ATTRIBUTE[Fuse.COUNTY]="Oxford";
```

ClosestServiceAddress Table

The optional ClosestServiceAddress table is used to store information for a specific custom spatial query: determining the closest service address to a point feature. This spatial query will add the first retrieved service address from the spatially closest transformer as an attribute to point features. However, to use this table, ensure the “SpatialAttributesForPoints” Boolean is checked (yes or true) in the “Defaults table, otherwise this table will be ignored.

This functionality assumes:

- that the transformer feature class has a key field for the premise
- the customer table has a corresponding key field for the premise
- a relationship exists between the transformer class and customer table
- the customer table exists/is accessible from the geodatabase.

Parameter	Value	Type
TransformerClassName	Name of the Transformer class to obtain the closest service address from (i.e. the feature class that serves as the origin in the premise relationship).	Text(50)
TransformerClassPremiseKey	Name of the field in the transformer class that serves as the key for the premise relationship.	Text(50)
CustomerTableName	Name of the SDE customer table that contains the service addresses.	Text(50)
CustomerTablePremiseKey	Name of the field in the customer table that serves as the key for the premise relationship.	Text(50)
AddressFields	The exact names of the customer table address fields that are desired in the attribute, in the exact order they are desired in.	Text(255)

Parameter	Value	Type
PremiseCompanyKey	Name of the Company field in Premise table	Text(50)
CustomerCompanyKey	Name of the Company field in Customer table	Text(50)

For example, a transformer class named “Transformer” contains a key field named “PremiseNumber”. This is the key in a class relationship to the customer table called “Customer” that has the key field “Premise_ID” located in the same geodatabase.

Meanwhile, the address fields in the “Customer table that are required for the service address attribute are “STREET_NUMBER”, “STREET_NAME”, “STREET_TYPE”, and “STREET_DIRECTION”. These fields need to be entered in the “AddressFields” field in comma delimited form, ensuring the field names as exactly as they appear in the “Customer” table, and in the exact order they are desired in.

Such an example would look like the following:

Parameter	Value
TransformerClassName	“Transformer”
TransformerClassPremiseKey	“PremiseNumber”
CustomerTableName	“Customer”
CustomerTablePremiseKey	“Premise_ID”
AddressFields	“STREET_NUMBER, STREET_NAME, STREET_TYPE, STREET_DIRECTION”
PremiseCompanyKey	“COMPANY”
CustomerCompanyKey	“COMPANY_CD”

As an example, this spatial query may result in the following attribute being added to a fuse feature in the extract (mp) file:

```
ATTRIBUTE[Fuse.CLOSEST_SERVICE_ADDRESS]="1598 Maple Ave E";
```

Explicitly Specifying Data Ownership

Personal geodatabases are not affected by this functionality.

For the tables referenced in the Reference Database Tables section, if the SDE database UserID that owns the data is different from the UserID used for performing the extract, then the data-owning UserID must be explicitly specified as a prefix for items such as class, dataset, and version names. This is necessary because SDE interfaces to Oracle Utilities Network Management System, and the standard method of explicitly declaring data ownership in Oracle Utilities Network Management System is <owner_dbusername>.<item>.

For example, if the Adapter is being run as SDE user ‘abc’ and the data is owned by user ‘xyz’, then any reference to class, dataset, or version items must be prefixed according to the Oracle Utilities Network Management System method. Several examples of such item definitions are provided:

Parameter	Value
Version	“xyz.MAPPING_DEFAULT”
DataSet	“xyz.Electric_Feeder”
ClassName	“xyz.DistributionTransformer”

Freeze Versions Created and Used by the Adapter

The Adapter creates and maintains “freeze” versions when a versioned database is available and the “UseSDEVersioning” attribute is checked/true in the “Defaults” table of the application reference database. These versions are used to minimize the time the Adapter locks the database as well as support change detection.

Personal geodatabases and unversioned databases are not affected by this functionality. Extracts of unversioned databases are automatically conducted on the default database. No special action is required.

If the database is registered as versioned and the user has configured the Adapter to use SDE versions, the Adapter generates a temporary freeze of the desired extract version once the extraction begins. The extract is then carried out on the freeze version, which makes the existing “default” version available for other purposes (e.g., new data postings) while the extract is running.

Then, once an extract successfully completes, the temporary freeze is renamed to a permanent freeze. This permanent freeze version must be kept because it can be used in subsequent change detections as a static representation of the database at the time of the last extract (i.e. as a temporal benchmark). In other words, it maintains a handle on database state as it existed when the last extract took place.

Several benefits of this approach include:

- A freeze can always be referenced back to a single base (parent) version of data.
- New work version (i.e. posting and reconciling) can continue on the “as-built” or current version concurrently with an extract process.
- Data should never change on the freeze version, and therefore the Adapter should never be “locked out” of the database.
- The freeze can be re-used as part of change detection tool prior to the next extraction in an effort to determine which features and objects changed since the last extract.

When the Adapter is launched against a versioned database, all version information is loaded to the Adapter. The Adapter then identifies the "current" or "extract" version as defined in the application reference database.

If a non-zero value is placed in the "NumDaysToKeepFreeze" field of the "Defaults" table of the application reference database, each freeze version is checked to determine if it has "expired." If a version is older than the maximum specified, then the version is automatically dropped. All remaining freeze versions are then listed in the change detect field, however the most recent freeze should be automatically selected as the default.

Once an extract commences, a temporary "freeze" (copy) is created of the extract version. The extract is then run on the freeze version, effectively freeing the extract version for other purposes. As a result, it is possible to simultaneously post new data to the "current" version of the database while an extract is in progress.

Freeze Naming Conventions

The naming convention is to prefix the valid freeze version with "Freeze_" and suffix with the work order name for the extract process. In this manner, valid freezes can be identified "at-a-glance" while their specific purpose is also revealed. Of course, the description attribute for the version is also populated to reflect the purpose of the freeze.

Temporary or working freezes are prefixed with "Extract_" and suffixed with a date stamp. These versions are for internal purposes only, and should be automatically cleaned up (removed) by the Adapter when it is launched after a few days.

Managing Freezes

To optimize the database, versions should be managed. The administrator can specify in the application reference database (in the "NumDaysToKeepFreeze" field of the "Defaults" table) the number of days that valid freeze versions should be kept. Past that date, freezes will be automatically deleted on the next launch of the Adapter. Note that a value of zero (0) in this field means "keep the freeze versions indefinitely."

At their discretion, a system administrator can manually delete freeze versions and compact the database. However, be aware that if the latest freeze version is deleted, past state information will be lost and there will be no way to detect database changes since the last successful extract. In other words, the Adapter's change detect tool will not work the next time an extract is required. In this case, it would be up to the GIS maintenance team to manually determine the appropriate partitions for extraction.

Running the Adapter

The Adapter can be run either in the background with command line parameters or interactively with a GUI. In any event, ensure the application reference database file has been customized prior to first running the Adapter.

Launching Adapter with Command Line Parameters

The Adapter can be launched in a non-GUI mode from the Windows command line with parameters. However, because this launch method is "silent," no visual feedback will usually be provided to the workstation screen except for error reports. The user will have to view the output directory in order to determine progress.

As a result, the Windows scheduler can be used to schedule an automatic extract process. Instructions on configuring and using the scheduler can be obtained through Microsoft's Help files, or through the scheduler wizard (available in Windows 2000).

When launching the Adapter in non-GUI mode, only the non-GUI launch flag is required, while all other command line parameters are optional. If a parameter is omitted, the default is assumed. Note the default dataset or partition parameters that will be used should be already defined by the user in the "Defaults" table of the application reference database.

The command line usage is:

```
"[install directory]\AItoCentricity.exe" -nongui [-?] [-d dataset] [-r
reference] [-p partition] [-w workorder]                    [-o] [-
e extract] [-q queue]
```

Option	Description
-nongui	(required) Indicates to launch Adapter in non-GUI mode.
-?	Command Line Help
-d dataset	Dataset(s) to process; separate with commas (e.g., -d electrical,landbase)
-r reference	Path of application reference database file (e.g., -r c:\myrefdb.mdb)
-p partition	Partition type to use; as in PartitionReference table (e.g., -p Electric_Tile)
-w workorder	Workorder directory to create (e.g., -w Nov20-01)
-o	Force an Overwrite of existing workorder and version (if existing ones are found)
-e extract	Extract type to perform; 'FACILITY' for facility data (default) 'NON' for non-facility data (i.e. landbase) (e.g., -e NONFACILITY)
-q queue	Method to queue partitions; 'ALL' queues all partitions (e.g., -q ALL) 'CD' uses change detection to queue (e.g., -q CD) (default) Specifying a valid (existing) selection set name will queue the partitions identified in the partition queue table for that selection set (e.g., -q 'WeeklyLandbaseExtract')

Launching Adapter Interactively

The AItoCentricity program can be launched from the Windows Start button Program – SPL' AItoCentricity submenu. As it starts, a splash screen appears.

After a few moments, the program will then configuration screen for the ArcGIS adapter. This screen contains four main framed panels. The top three are categories of information:

- Extract Data
- Change Detection
- Data Extraction Parameters

Across the bottom of the GUI is the fourth framed panel, the Status panel, where information on the process is reported to the user. Other objects include the work order field, and the extract and cancel buttons.

Extract Data Panel

This panel contains information on the Extract version (as defined in the application reference database). This should be the "current" or "proposed" data version for the OMS, and it should contain the modified or most recent data since it is the version that objects and features will be extracted from.

Change Detection Panel

This panel contains information for the built-in change detection functionality of the Adapter. If the SDE database is versioned and the "UseSDEversions" field of the Defaults table is checked, then this panel will become enabled. Otherwise, it should remain grayed out.

If the above criteria are met and the user wishes to perform a change detection, then a Change Detect version must first be chosen in the Version sub-panel. In order to detect changes in the database, the system must have some method of maintaining the database state on the last successful extract. Therefore, the change detect version represents a "check-out" or "freeze" of the data at the point the Adapter last successfully completed. By default, the system will attempt to choose the most recent data freeze, based on creation date. If no previous freeze is detected, then the field is left blank. Note that freeze version names are always prefixed with the string "Freeze_".

It is possible that a user may want to choose a different version for change detection. Even though the most recent freeze is selected, the drop down list also provides all of the available previous data freezes. Meanwhile the optional ChangeDetectVersionName field in the Defaults table of the application reference database could be populated to override the automatic selection of the most recent freeze. This option is provided for convenience only, and this approach is not recommended unless the potential impact of using a change detect version not created by the system is clearly understood by the user.

The user can also specify a unique name for the change detection results in the Save Results Set to a Table sub-panel. This is the name of the set that will be saved to the PartitionQueue table when the change detection completes.

Then, to have the Adapter perform change detection, click the Perform Change Detect button. This will automatically queue partitions that contain objects that are different in the extract version versus the change detect version (which is usually a freeze of the data on the last extract). Because of the intensive database search, reconciliation, and comparisons required for performing change detection, please be aware it may take a while for the Adapter to complete this task.

Data Extraction Parameters Panel

This panel contains the data parameters required for conducting the extraction.

First, partition information for the extract must be specified. For this, the partition type is selected from the Partition Type drop-menu. This is the type of partition to be used for the extract, as defined in the "PartitionReference" table of the application reference database.

Note that depending on the partition type chosen, the items (partition names) listed in the "Partitions" queue on the right hand side of the panel will change to reflect the unique partition names associated with the specified partition type.

Next, based on the partition type chosen, the associated datasets must be chosen for the extract. Selecting more than one dataset is allowed, but be aware that objects from all chosen datasets will end up in the same partition model preprocessor output (.mp) file.

One of three partitions selection modes also must be specified: Manual Selection, Filter by Date, or Filter by Partition Set.

In Manual Selection mode, the user may manually choose/queue which partitions to extract from the Partitions list on the right hand side of the GUI. Note that clicking or highlighting a partition on the Partitions list will queue it for extraction. Standard Windows shift-click or alt-click functionality is employed for multiple selects.

In Filter by Date selection mode, the user can specify dates for filtering the selections or partitions by clicking on the From Date or To Date buttons and interactively double-clicking a date on the calendar control. Then, to apply the date filter and queue the appropriate partitions, the user must click the "Select Partitions by Filter" button.

In "Filter by Partition Set" selection mode, the user can specify a pre-defined or saved queue or set of partitions by selecting the name of the selection set from the list. Then, to apply the partition set filter and queue the appropriate partitions, the user must click the Select Partitions by Filter button.

Other tools include the Select All and Unselect All buttons, which will select all the partitions in the queue, or none of them respectively.

Finally, if the PartitionQueue is not marked as ReadOnly in the application reference database, the user can choose to name and save the currently selected queue of partitions to a set for later use. To do this, enter a unique name in the Name of Partition Set field and click the Save button.

Extract Launch Controls

In order to identify the extract results, the user must specify a work order name for the extract. The default name for an extract is a combination of the system's current date and time (e.g., 2001Aug30-1256), however this can be overridden by simply changing the contents of the Work Order Name field.

Pressing the Launch Extract button begins the extraction of the specified items. Once the extract begins, the Exit button in this panel is replaced with a Cancel button, which if clicked will allow the user to interrupt the extract process.

Status Panel

Because the extraction process will take time to complete, its status and progress can be monitored from the panel at the bottom of the GUI. The Overall Progress field displays a very rough estimate of the entire work order process (i.e. all feature classes for all datasets within all queued partitions for the extraction), while the Detail Progress field indicates more detailed progress information, such as the specific feature class currently being processed.

Populate the Required Relationship Tables

The RequiredRelationships table of the Adapter's configuration database file AItoCentricity.mdb is used to indicate which relationship between classes and objects in the GIS database should be traversed and extracted. It could be cumbersome to manually identify every available feature relationship in the ArcGIS database and populate this table. Using this SQL query to the GIS database, you can get a listing of the relationships in the GIS model:

```
SELECT GDB_FeatureDataset.Name as DatasetName,
    GDB_ObjectClasses.Name as Parent,
    GDB_ObjectClasses_1.Name as Child ,
    GDB_RelClasses.ForwardLabel as Label1 ,
    GDB_RelClasses.BackwardLabel as Label2
FROM (GDB_RelClasses
    INNER JOIN ((GDB_ObjectClasses
    INNER JOIN GDB_FeatureClasses
    ON GDB_ObjectClasses.ID = GDB_FeatureClasses.ObjectClassID)
    INNER JOIN GDB_FeatureDataset
    ON GDB_ObjectClasses.DatasetID = GDB_FeatureDataset.ID)
    ON GDB_RelClasses.OriginClassID = GDB_ObjectClasses.ID)
    INNER JOIN GDB_ObjectClasses AS GDB_ObjectClasses_1
    ON GDB_RelClasses.DestClassID = GDB_ObjectClasses_1.ID
ORDER BY GDB_FeatureDataset.Name, GDB_ObjectClasses.Name;
```

The query results with relationship information, such as Origin Class and destination class and lables, and these can be cut/pasted into the RequiredRelationship table.

Usage Scenarios

This section presents several usage scenarios for the AItoCentricity data Adapter.

- **Manually Selected Partitions (no change detection)**
- **Save and Recall a Partition Queue**
- **Load Partition Queue from Externally Generated Source**
- **Bypass Use of the ESRI ArcSDE Database Versions**

Manually Selected Partitions (no change detection)

If no table name is specified in the Partition Queue Reference table, the user will not be able to perform Change Detection and will be restricted to a manual selection of the Partitions.

The following portions of the GUI will be disabled:

- Change Detection panel
- Partitions Queue Selection Mode panel
- (Table Name label will be replaced with No Partition Queue table and the options will be locked to Manual Selection)
- Select Filtered Partitions button
- Save Selection to Table section

In this scenario, the user selects a combination of Partition Type/Extract_Type (e.g., “Electrical *.mp”) from the Partition Type drop-menu. The application queries table *DatasetPartitionTypeExtractType_Relation* in order to get the related Data Sets and populate the Data Sets field.

The application also uses the Partition Type selection in order to query ArcGIS SDE and populate the Partitions field with the related Partition Names. The user manually selects the Partitions.

The user then specifies a Work Order Name and launches the Adapter by clicking the Launch Extract button. The extraction is performed using the selected Partition Type, the listed Data Set(s) and the selected Partition(s).

Please note that all listed Data Set(s) are being used (no selection available) since the information is now contained in configuration table *DatasetPartitionTypeExtractType_Relation*.

Save and Recall a Partition Queue

This scenario corresponds to the case where change detection has been run, and the results must be saved and recalled at a later time.

In the two scenarios below, the user must ensure that a Table Name and a Connection Type are specified in *PartitionQueueReference* table and that the ReadOnly field is not selected. The default table specified in *PartitionQueueReference* is *PartitionQueue* located in AItoCentricity.mdb. If the specified table does not exist, the application will generate an error upon launch.

When the application is launched, the valid table name is specified within the Partition Queue Selection Mode panel. The Adapter queries the *PartitionQueue* table and populates the FromDate field with the earliest date and the ToDate field with the latest one. The Partition Set Names field is populated with the result of another query that selects all distinct Partition Set Names.

Perform Change Detection Via the Adapter

In the Change Detection Panel:

The user selects a freeze version (Static Data) from the drop down list and specifies a Name of Change Set. When the user clicks the Perform Change Detect button, the following happens:

- The GUI compares the selected "Change Detect Version" with the "Extract Version";
- The software reconcile both versions to get the affected SDE tables;
- For each table, the application will:
  - Get the geometry of all changed features
  - Find the dataset corresponding to the table
  - For each partition type corresponding to the dataset:
    - spatial select to find partitions (tiles)
    - store the result in a object or dictionary;
- The software files out the objects or dictionary into the *PartitionQueue* table.
- The software queries the *PartitionQueue* table WHERE "Partition Type" = *GUI selection (truncated) AND* "DataSet Name" = *listed Data Set(s)* AND WHERE "Partition Set Name" = *specified "Name of Change Set"*. The resulting Partitions are then highlighted.

Each record of the *PartitionQueue* table is a distinct PartitionName/PartitionType/ DataSet combination e.g., if one Partition has two related Data Sets, two separate records will be saved to the table. The Name of Partition Set field gets populated with the specified Name of Change Set and the Date Stamp field gets populated with the time when the result of the change detection (if successful) was saved to the table.

> **Note:** By saving a queue of the changed partition names, the next steps can be executed later on, even after a shut down and re-launch of the Adapter.

In the Data Extraction Parameters Panel:

- As in section 4.1, the user selects a combination of Partition Type/Extract_Type that will affect both the Data Sets and the Partitions listbox. In this scenario, the Partition Type selection will also be used as a parameter to query the *PartitionQueue* table and populate the Partition Set Names field with the resulting Partitions Sets.
- The user then selects the Filter by Partition Set option, which will enable the filed and the Select Filtered Partitions button. After highlighting one of more Partition Set Names, the user clicks the Select Partitions by Filter button. This command queries table *PartitionQueue* WHERE "Partition Type" = *GUI selection (truncated) AND* "DataSet Name" = *listed Data Set(s)* AND WHERE "Partition Set Name" = *GUI selection(s)*. The resulting Partitions are then highlighted. Please note that the filtered partitions can only be related to one Partition Type at a time.
- The user can now perform the extract as usual. However, in the case of a successful extract, if one or more Partition Set is highlighted under Filter by Set Names, the Extract Date will be saved for each associated record in *PartitionQueue* table.

Manually Select Partitions and Save Selection to a Partition Set

As above, the user selects a combination of Partition Type/Extract_Type, which will affect both the Data Sets and the Partitions field, and manually selects the Partitions. Note that in this scenario, in order to be able to manually select the Partitions, the user must ensure Manual Selection in the Partition Queue Selection Mode panel is specified.

- The user decides to save the selection and wait before performing the extract. Under Save Selection to Table the user specifies a Partition Set Name and clicks the Save button. This new Partition Set is saved to the *PartitionQueue* table and is automatically added to the Partition Set Names list in the GUI.

- The user shuts down the application and re-launches it later to perform the extract on the selection. The user must first select the proper Partition Type i.e. the same "Partition Type" as when the selection was saved, otherwise, the Partition Set will not be displayed in the Partition Set Names list.
- The user then selects the Filter by Set Name option, highlights the Partition Set and clicks the Select Partitions by Filter button to query the *PartitionQueue* table and to highlight the resulting Partitions.
- The user can now perform the extract as described above.

A variant of this scenario is for the user to first automatically select partitions by using the Filter by Partition Set option as described in above. They can then click on the Manual Selection option which keeps the highlighted Partitions and allows the user to select or de-select partitions when holding the "Control" or the "Shift" key. The user can now perform the extract or save his new selection. Note that when specifying Manual Selection, the Partitions are still highlighted but the Partition Set Names are de-selected – this identify the selection as manual and therefore no Extract Date will be saved during the extract.

Load Partition Queue from Externally Generated Source

As above, *PartitionQueueReference* table must be populated with the appropriate information.

- The user selects a combination of Partition Type/Extract_Type, which will affect both the Data Sets and the Partitions listbox. The user can then follow the steps described in section 4.2.1, under "In the Data Extraction Parameters Panel".
- Another option that the user has is Filter by Date. With this option, the user selects the FromDate and ToDate, using a calendar control that restricts the format. The user then clicks the Select Partitions to Filter button. This commands queries table *PartitionQueue* WHERE "Partition Type" = *GUI selection (truncated)* AND "DataSet Name" = *listed Data Set(s)* BETWEEN "FromDate" AND "ToDate". The resulting Partitions are highlighted.
- The user can now launch the Extract. If the extract is successful and if table *PartitionQueue* is in Read/Write mode (i.e. ReadOnly is unchecked in *PartitionQueueReference*), the Extract Date field of table *PartitionQueue* could be updated by the following statement "UPDATE "Extract Date" WHERE "Partition Type" = *populated object AND* "DataSet Name" = *populated object* AND "Partition Name" = *populated object* BETWEEN objFromDate AND objToDate.

Bypass Use of the ESRI ArcSDE Database Versions

By unselecting field "Keep versions" of table *Defaults*, the Adapter will not generate ArcSDE database versions when it is launched. In choosing this option, the Change Detection cannot be performed and will therefore be disabled in the GUI.

Populating the Partition Listbox Using PartitionQueue Table

The Adapter Graphic User Interface populates and displays all available partitions in the Partition field. These partitions are read from the specified Partition class in the ArcGIS. Due to Visual Basic software limitations, it could take an unusually long time to populate the Partitions field when there are more than 20,000 partition names. The time to partitions name could vary from few minutes to hours based on the network condition and database server hardware limitation When the Adapter is used in command line mode, the number of partitions does not cause any performance degradation but in interactive mode, a large number of partitions affect the Adapter GUI performance.

Read Predefined Partition Queue from PartitionQueue Table

This option assumes that change detection has been performed prior to the Adapter being launched, and that the results are available for the Adapter to read. Usually, this approach means that change detection is performed either as part of data maintenance, or as a batch job prior to extraction.

Out of the scope and control of the Adapter, the results of external change detection are placed in a table (with specific schema e.g., PartitionQueue) in the database. The Adapter is then configured to “read” the unextracted partitions from this table, and populate the Partition Queue field in the Adapter GUI. The partition is considered “unextracted” when the extract_date field is null. After successful extraction of each partition, the extract updates this field with date and time stamp indicating that partition has been extracted.

Note that maintenance of the externally-generated change detect or the partition queue table is out of the scope of the Adapter, and alternate tools must be in place to manage the data (i.e. the partition queue table).

The users can define an external table named PartitionQueue to save a list of partitions to be extracted.

Partition Loading

The user-configurable option in the configuration database will determine whether the Adapter should populate the Partition field with all available partitions in a specified PartitionClass or use PartitionQueue table to populate the Partitions field.

Load All Partitions

If UsePartitionQueue is unchecked in the Default table, then the Adapter will populate the Partition field will all available partitions for a selected partition class. In this scenario, the user selects a combination of Partition Type/Extract_Type (e.g., “Electrical *.mp”) from the “Partition Type drop-menu. The application uses the Partition Type selection in order to query the database and populate the Partitions field with the related Partition Names.

Load Externally Generated Partition Queue Saved in PartitionQueue

If UsePartitionQueue is checked in the Default table then Adapter will not populate Partition field all of the available partitions for a selected PartitionClass. Instead, the field will be populated using all “unextracted” partitions in the PartitionQueue table.

The user must ensure that a Table Name and a Connection Type are specified in *PartitionQueueReference* table and that the ReadOnly field is not selected. The default table specified in *PartitionQueueReference* is *PartitionQueue* located in AItoCentricity.mdb. If the specified table does not exist, the application will generate an error upon launch.

When the application is launched, the valid table name is specified in the Partition Queue Selection Mode panel. The Adapter queries the *PartitionQueue* table and populates the FromDate field with the earliest date and the ToDate field with the latest one. The Partition Set Names field is populated with the result of another query that selects all distinct Partition Set Names. The user selects a combination of Partition Type/Extract_Type, which will affect both the Data Sets and the Partitions fields.

Filter By PartitionSet

As described in a previous section the user selects a combination of Partition Type/Extract_Type that will affect both the Data Sets and the Partitions fields. In this scenario, the Partition Type selection will also be used as a parameter to query the *PartitionQueue* table and populate the Partition Set Names field with the resulting Partitions Sets.

The user then selects the Filter by Partition Set Names option, which will enable the field and the Select Partitions by Filter button. After highlighting one or more Partition Set Names, the user clicks the "Select Partitions by Filter button. This command queries table *PartitionQueue* WHERE "Partition Type" = *GUI selection (truncated) AND* "DataSet Name" = *listed Data Set(s)* AND WHERE "Partition Set Name" = *GUI selection(s)*. The resulting Partitions are then populated in the Partitions field. Please note that the filtered partitions can only be related to one Partition Type at a time.

The user can now perform the extract as usual. However, in the case of a successful extract, the "Extract Date" will be saved for each associated record in the *PartitionQueue* table

Filter By Date

Another option that the user has is Filter by Date. With this option, the user selects the FromDate and ToDate, using a calendar control that restricts the format. Clicking the Select Partitions by Filter button queries table *PartitionQueue* WHERE "Partition Type" = *GUI selection (truncated)* AND "DataSet Name" = *listed Data Set(s)* BETWEEN "FromDate" AND "ToDate". The resulting Partitions are populated in the Partition field.

The user can now launch the Extract. If the extract is successful the "Extract Date" field of table *PartitionQueue* is updated by the following statement "UPDATE "Extract Date" WHERE "Partition Type" = *populated object AND* "DataSet Name" = *populated object* AND "Partition Name" = *populated object* BETWEEN objFromDate AND objToDate.

Output of Adapter

The model preprocessor import files (.mp), the log file, and the work order time stamp file are all placed in a folder with the work order name. This folder will appear in the output directory as designated in the "Defaults" table. If change detection was run, a log file of the process will be placed in the root of the output directory.

Note that if the extract is still in progress or does not complete, the work order time stamp file will not appear in the output directory. The work order time stamp file will only appear if the complete extract finishes successfully. As such, external processes can use its presence as an indicator of a completed extract.

Model Preprocessor File

The main products of the Adapter are the Model Preprocessor (".mp") files. There is one preprocessor file for each partition.

A sample of a model preprocessor file that contains a switch with feature linked annotation and a related SwitchUnit is shown below:

```
ADD Switch 97 {
PORT_A=4.239.1;
PORT_B=4.240.2;
DIAGRAM[point] {
HEIGHT = 0;
ANGLE = 45;
SCALE = 1;
GEOMETRY=
(401999.99995809,5000249.99990163);
};
DIAGRAM[Switch_Anno] = {
HEIGHT = 7;
ANGLE = -44.9999894816408;
SCALE = 1;
TEXT = "SW0097";
JUSTIFICATION = LL;
GEOMETRY = {
```

```
(401999.99995809,5000249.99990163);
};
ATTRIBUTE[Switch.FEATURECLASSID]="47";
ATTRIBUTE[Switch.OBJECTID]="97";
ATTRIBUTE[Switch.SUBTYPE]="Fused Cutout/Switch";
ATTRIBUTE[Switch.FACILITYID]="Applebee Junction";
ATTRIBUTE[Switch.SYMBOLROTATION]="45";
ATTRIBUTE[Switch.FEEDERNUMBER]="F159-J1";
ATTRIBUTE[Switch.SCADA]="No";
ATTRIBUTE[Switch.OPERATINGVOLTAGE]="14,400 kva";
ATTRIBUTE[SwitchUnit.OBJECTID]="455";
ATTRIBUTE[SwitchUnit.AMPRATING]="600 Amps";
ATTRIBUTE[SwitchUnit.MANUFACTURER]="Westinghouse";
ATTRIBUTE[SwitchUnit.PHASEDESIGNATION]="C";
ATTRIBUTE[SwitchUnit.SUBTYPE]="Switch Unit with Fuse";
ATTRIBUTE[SwitchUnit.FACILITYID]="Applebee Lane";
ATTRIBUTE[SwitchUnit.INSTALLATIONDATE]="00/03/19";
ATTRIBUTE[SwitchUnit.SERIALNUMBER]="SU_R569-CX89865";
ATTRIBUTE[SwitchUnit.MANUFACTUREDATE]="99/07/27";
ATTRIBUTE[Switch_Anno.FEATURECLASSID]="88";
ATTRIBUTE[Switch_Anno.OBJECTID]="168";
ATTRIBUTE[Switch_Anno.FEATUREID]="350";
ATTRIBUTE[Switch_Anno.ELEMENT]="BLOB";
ATTRIBUTE[Switch_Anno.SHAPE.AREA]="2928.31356";
ATTRIBUTE[Switch_Anno.SHAPE.LEN]="248.915904366188";
};
```

Extract Log File

The diagnostic product of the Adapter is a log file (“extracting.log”).

A sample extract log file may look like this:

```
===============================================
 Process Log File
 ----------------
 ArcGIS to ORACLE NMS Data Extraction Utility for Work Order:
2001Dec30-1258

 Generated By: AItoCentricity  ver: 3.0.0
 File Name:    c:\SPL\2001Dec30-1258\extracting.log
 Date Opened:  01/12/30 12:58:07
===============================================

01/12/30 12:58:07
01/12/30 12:58:07            Creating temporary freezes for database
version OMS Proposed
01/12/30 12:58:07            Database temporary freeze
Extract_Freeze_2001Dec30-1258 created for maintaining state.
01/12/30 12:58:07            Database temporary freeze
Extract_010830125807 created for extracting from
01/12/30 12:58:07            Open temporary freeze for processing
01/12/30 12:58:08                Server:    gis_server
01/12/30 12:58:08                Database:
01/12/30 12:58:08                Instance:  esri_sde
01/12/30 12:58:08                Version:   Extract_010830125807
01/12/30 12:58:08            Class relationships obtained for dataset
Electric_Points
01/12/30 12:59:03            Class relationships obtained for dataset
Electric_Lines
01/12/30 12:59:03
01/12/30 12:59:03            Partition Type: Electric_Partitions
```

```
01/12/30 12:59:03          Extract Method: Spatial
01/12/30 12:59:04          ________________________________________
01/12/30 12:59:04
01/12/30 12:59:04         Partition 754490 as file c:\SPL\2001Dec30-
1258\754490.mp
01/12/30 12:59:04
01/12/30 12:59:04           * DataSet:   Electric_Feeder
01/12/30 12:59:04
01/12/30 12:59:04          ConductorJunction
01/12/30 12:59:06              1 - point
01/12/30 12:59:06                - is networked
01/12/30 12:59:07                - feature linked annotation
01/12/30 12:59:07              2 - point
01/12/30 12:59:07                - is networked
01/12/30 12:59:08                - feature linked annotation
01/12/30 12:59:08              3 - point
01/12/30 12:59:08                - is networked
01/12/30 12:59:09                - feature linked annotation
01/12/30 12:59:09              4 - point
01/12/30 12:59:09                - is networked
01/12/30 12:59:10                - feature linked annotation
01/12/30 12:59:10              5 - point
01/12/30 12:59:10                - is networked
01/12/30 12:59:11                - feature linked annotation
01/12/30 12:59:11            ElectricLineSegment
01/12/30 12:59:12              1 - line
01/12/30 12:59:12                - is networked
01/12/30 12:59:12              2 - line
01/12/30 12:59:12                - is networked
01/12/30 12:59:13            PrimaryMeter
01/12/30 12:59:13              1 - point
01/12/30 12:59:13                - is networked
01/12/30 12:59:14                - related tabular attributes
```

Work Order Time Stamp File

Once the work order extract successfully completes, a zero byte file is created in the extract subdirectory to indicate the time of completion ("WO_TIME_STAMP").

Change Detect Log File

The diagnostic product of the change detect mechanism is a log file. These log files are placed in the "root" output directory, as specified in the ExtractDirectory field of the Defaults table of the application reference database. They are prefixed with "ChangeDetect_" and have a date stamp suffix, where the date stamp format is YYMMDDHHMMSS.

A sample change detect log file may look like this:

```
===============================================
 Process Log File
 ----------------
 ArcGIS to ORACLE NMS Data Extraction Utility Change Detect Log File

 Generated By: AItoCentricity  ver: 3.0.0
 File Name:    c:\SPL\ChangeDetect_011217141538.log
 Date Opened:  01/12/17 14:15:38
===============================================

01/12/17 14:15:38          ***** CHANGE DETECT LOG FILE *****
01/12/17 14:15:39
01/12/17 14:15:39          ----------------------------------------
01/12/17 14:15:39          Change Detect For Dataset Electric_Feeder
```

```
01/12/17 14:15:39
01/12/17 14:15:39            *** Obtain Unique DB Connection
01/12/17 14:15:39
01/12/17 14:15:39           *** Set versioned workspace to the opened
workspace
01/12/17 14:15:39
01/12/17 14:15:39            *** Reconcile the Versions
01/12/17 14:15:39            Reconciling OMS Proposed with
Freeze_2001Dec15-1532
01/12/17 14:15:39
01/12/17 14:15:39         Checking DISTRIBUTIONTRANSFORMER class for
changes . . .
01/12/17 14:15:40                - Modified OID=1362
01/12/17 14:15:40                  - change detect partition(s):
01/12/17 14:15:41                    758490-A
01/12/17 14:15:41                  - extract partition(s):
01/12/17 14:15:41                    758490-A
01/12/17 14:16:00
01/12/17 14:16:02            Checking UGELECTRICLINESEGMENT class for
changes . . .
01/12/17 14:16:12                - Modified OID=255
01/12/17 14:16:12                  - change detect partition(s):
01/12/17 14:16:12                    758490-A
01/12/17 14:16:12                    754316-B
01/12/17 14:16:12                  - extract partition(s):
01/12/17 14:16:13                    758490-A
01/12/17 14:16:13                    754316-B
01/12/17 14:16:23
01/12/17 14:16:23            Checking BUSBAR class for changes . . .
01/12/17 14:16:24                - Modified OID=1879
01/12/17 14:16:24                  - change detect partition(s):
01/12/17 14:16:24                    758490-A
01/12/17 14:16:24                  - extract partition(s):
01/12/17 14:16:25                    758490-A
01/12/17 14:16:26
01/12/17 14:16:26            ----------------------------------------
01/12/17 14:16:26            *** Identifying partitions containing
modified features
01/12/17 14:16:28              754316-B
01/12/17 14:16:28              758490-A
01/12/17 14:16:28
01/12/17 14:16:28              Queued 2 partitions for extraction.
01/12/17 14:16:28
01/12/17 14:16:28            Change Detection Complete

===============================================
 Date Closed: 01/12/17 14:16:28
===============================================
```

Performance

Performance of the extraction is currently at "GIS speeds". Landbase data does not typically have connectivity issues so extracts can take as few as several seconds per typical partition. However, complex Electrical networks may take about 5-10 minutes per typical partition to extract.

The number of class relationships to traverse when extracting objects will dramatically affect Adapter performance. Therefore, the RequiredRelationships table in the application reference database should be configured to only allow relationships vital for the model build process. This in turn will also affect the size of the output files.

The depth of related features and objects that the extract traverses (set in the Defaults table of the application resource database file) can also dramatically impact performance, so keep these numbers to the minimum possible.

The time required to perform change detection will vary and is dependent on the number of features and objects that have changed since the last extract. Objects are non-spatial and the system must therefore branch out from objects to their related features in order to obtain an approximate geographic location. Therefore, as a rule, objects are much more time-intensive to perform a change detection on than features.

The performance is severely degraded when spatial attribute queries are performed.

Issues

Windows Resources

The Adapter uses the ArcObjects API. Currently, there are memory leaks either in ArcObjects or Visual Basic (Microsoft OS) that consume GDI resources. Typical symptoms of these resource limits being reached are: screen degradation, especially when a screen saver application is running, changing of the down arrow icons on the application GUI to a digit such as "6", or complete failure of the operating system once the Adapter program has completed and exited (requiring a reboot).

> **Note:** The Adapter minimizes the use of GDI resources as much as possible, but the Windows OS limit could potentially be reached if the Adapter is run against many partitions in a single session.

Chapter 5

Intergraph G/Electric Adapter

This chapter includes the following topics:

- **Introduction**
- **Adapter Capabilities**
- **Programs and Files**
- **Installing the G/Electric Data Adapter**
- **Adapter Configuration**
- **Third Party Products**
- **Executing the Adapter**
- **GIS Data Requirements**
- **Uninstalling the G/Electric Data Adapter**

Introduction

GIS to Oracle Utilities Network Management System integration is a multi-step process that generates an operational topological representation of the existing GIS database for use by and display in the Oracle Utilities Network Management System. This process is referred to as the model build. A single spatial grouping of GIS objects (known as a partition) is processed by the Oracle Utilities Network Management System model build engine before incorporating the partition data into the Oracle Utilities Network Management System electrical model. An electrical circuit (feeder) or polygon feature, such as a service territory boundary, can be used to define a partition. The Oracle Utilities Network Management System model build engine processes partitions in three stages before committing it to the Oracle Utilities Network Management System operational electrical model: 1) GIS data extraction, 2) model pre-processing and 3) data import into Oracle Utilities Network Management System.

The G/Electric Adapter extracts geospatial data in Oracle's vendor-neutral model preprocessing file format (MP file). Geospatial data in G/Electric is stored in a number of geographic data and attribute data tables. The GIS stores and renders the geospatial data as a contiguous map. For performance and architectural reasons, Oracle Utilities Network Management System requires GIS data to be brought into the system as smaller chunks referred as partitions.

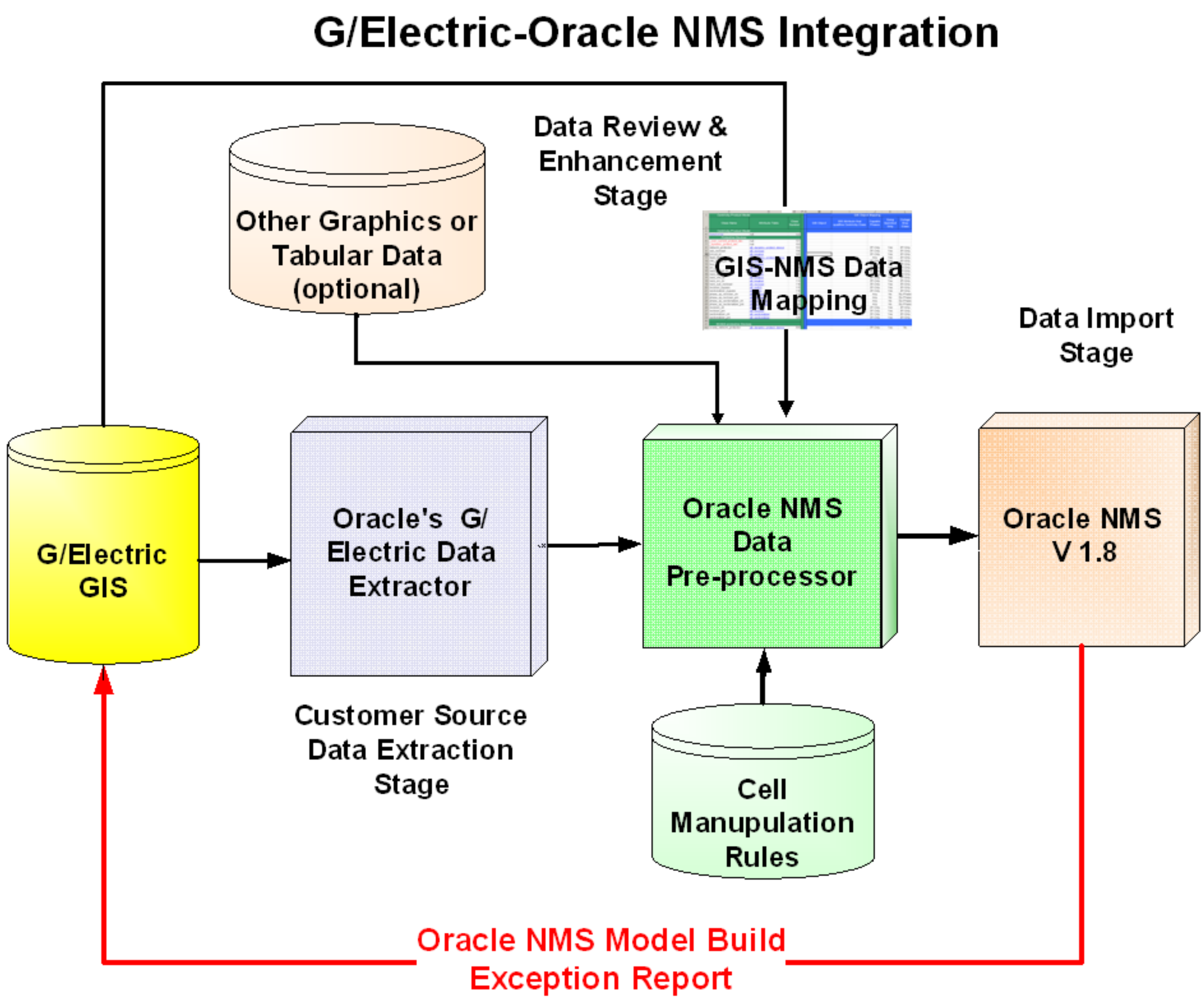
An electrical circuit (feeder) or polygon feature, such as a service territory boundary can be used to define a partition. Land base features must be separated from electrical features because this data is inherently different, and because land base data is typically more static.

Oracle's G/Electric Adapter is able to:

- Extract the data by predefined partitions.

- Extract the electric facility data by feeder; i.e. when a feeder is extracted, all the devices that take part in forming the feeder (i.e., are electrically connected) will be extracted into a single MP file.
- Extracts land base features by predefined polygon boundary in a single MP file.

In order to minimize the data extraction and model build data volume, the extract is capable of identifying the changed partitions since the last data extract operation.



Adapter Overview

The Adapter makes use of G/Electric GIS metadata to access the G/Electric GIS features. The Adapter utilizes default values as set in an accompanying Microsoft Access database, which has the same name as the executable (GElec2Oracle) and resides in the same subdirectory. This required database is referred to as the “application configuration” database.

The Adapter can be installed and run on any personal computer with Windows XP or Windows 2000 as the supported platforms.

There are two methods of invoking the Adapter:

- Non-interactively by a command line with parameters
- Interactively with a Graphical User Interface (GUI)

The program can be launched in non-GUI mode from an operating system command prompt with command line parameters. The user can either use the defaults as set in the application configuration database or override them with passed parameters. As such, the program can be scheduled for "silent" launch as a Windows scheduled task. Note that because this type of launch is "silent" and does not make use of a GUI, feedback is usually not provided to the workstation screen unless an error occurs.

The program can be launched from the workstation desktop or pull down menu. Using the GUI, the user can interactively control what data is extracted, what extract method is to be used, the type of data to extract, and the name of the output directory. The user is also shown a status area, which includes progress bars and messages for monitoring the progress of the overall work order and the specific partition currently being extracted.

The remainder of this document describes the capabilities of the Adapter, how to install and configure it, and how to run it.

Adapter Capabilities

The Adapter is a partition-based tool that extracts objects and features as selected by their geographic location (i.e., based on the feeder name or partition to which they belong).

Optionally, the Adapter can be used to detect changes in a GIS database, thereby automatically flagging partitions for extraction. Automated change detection by the Adapter is based on the G/Electric modification log table where the Adapter attempts to identify changed feeders or partition polygons since the last successful extract (based on creation and modification dates). The change detect functionality is an optional automated mechanism for queuing partitions that contain objects and features that have been either changed, added, or deleted since the last extract.

On extraction, the Adapter retrieves:

- Partition details.
- Features. For each feature, the following are extracted:
    - Feature object class and identifier
    - Connectivity
    - Geometric (graphic) components including a feature annotation (text) component
    - Rows and attributes for all the non-graphic components of the feature, including annotation.

The Adapter produces output in a work order folder containing files in Oracle Utilities Network Management System Model Preprocessor (".mp") format and a log file of informational, warning, and error messages.

The details of these topics are discussed below.

Partition Details

The Adapter generates a "partition section" based on the geometry of the selection or tile. It uses the specified feature class as the class name for the partition and an attribute (e.g., "TileID") for the partition's name.

Feature Extraction

The Adapter generates Add records for each feature. The Add feature records list the feature, its topology, diagram records, and attribute records

Object Identifier

The Adapter uses the feature class name and feature ID to identify an object. For example, “Switch 24”. Note the feature ID is unique within the feature class.

Connectivity

The Adapter determines the connected (or networked) objects by analyzing the G/Electric connectivity component. The connectivity (topology) is established using the Node1 and Node2 ID attribute, a unique index number in the connectivity component.

The Oracle Utilities Network Management System AttTopoBuild program uses the GIS provided Port (Node) information to generate topology.

Geometric Information

The Adapter determines the geometry of the object and uses it to extract the coordinates scale, height, angle, and justification of the object.

Coded Value Look Up

In G/Electric there are no coded values, instead it provides the Value List for feature attributes for which user input should be limited to the system provided values. Because there are no coded values in the GIS database, the Adapter does not need to perform any coded value look up.

Names Containing Space Characters

It is possible for GIS feeder names or class user names to contain spaces (e.g., “Overhead Line). However, the Oracle Utilities Network Management System model build process does not recognize or accept class names with spaces, so a conversion must be made.

The accepted convention is that all spaces be substituted with underscores (e.g., “_”). Therefore, if source data class names have spaces in them, the final output model preprocessor (.MP) files will have underscores substituting the spaces. An example would be a class named “Overhead Line”. The ADD statement in the extract output file will show the name “Overhead_Line” instead.

Node Reduction

The interface program does not use a node reduction algorithm to reduce the number of nodes.

Partitions with Index Maps

The G/Electric GIS database is a continuous map. For performance reasons, Oracle Utilities Network Management System requires maps to be tiled, each tile corresponding to an Oracle Utilities Network Management System partition.

The electrical features are extracted by using the feeder names associated with a source device or listed in the Connectivity table. All the electric devices taking part in forming a feeder are extracted in single MP file. The Adapter also extracts electrical device owner features such as structure and poles in the same MP file. The feeder name is used to name the extracted MP file.

The spatial data extract requires a GIS polygon feature to extract the data. These polygon partitions can be any size and are not required to be uniform. The optimal sized partition polygon contains less than 5,000 objects (devices, background text, and background graphics). For example, in dense urban areas of the electrical network, smaller size partition polygons can be used. The Adapter can use one of the existing polygon boundary classes or users can define a new Oracle Utilities Network Management System specific polygon boundary class to define the partition polygon.

A unique text attribute (e.g., "GridID") of the polygon partition feature class will be used to name the extracted MP Files that will become the name of the Oracle Utilities Network Management System partition.

Programs and Files

The Oracle Utilities Network Management System G/Electric Data Adapter program, configuration database, help file, and other associated files, are supplied in a standard self-extracting Oracle Utilities Network Management System G/Electric Data Adapter Installer. The following files will be installed in the user-selected installation directory. Default installation directory is "C:\Program Files\OracleTUGBU\Oracle GElectric data Extractor".

- Oracle Utilities Network Management System G/Electric Data Adapter Program (GElec2Oracle.exe)
- Oracle Utilities Network Management System G/Electric read me file. (readme.rtf)
- Oracle_NMS_GElectric_Extractor_User_Guide.pdf
- Oracle_NMS_GElectric Extractor Installation Guide.pdf

The following reference files will be copied to the Reference subdirectory under the user-selected installation directory:

- Oracle Utilities Network Management System G/Electric Data Adapter On-line Help document (gelec2spl.chm)
- Oracle Utilities Network Management System G/Electric Data Adapter Configuration Database (GElec2Oracle.mdb)
- Oracle Utilities Network Management System G/Electric Data Adapter custom scrip to create the v_spl_fidfno View object (create_spl_view.sql)

G/Electric Data Adapter Installation Phases

There are four phases to installing the Adapter: application installation, Adapter configuration, Oracle client configuration, and testing:

- Install the G/Electric Data Adapter
- Populate the Adapter configuration database
- Configure the Oracle SQL*Net Client
- Test the Adapter

Minimum System Requirements

The PC is required to meet these minimum system requirements to install the G/Electric Data Adapter.

Processor	Pentium® III 800 MHz minimum
Operating system	Microsoft Windows 2000 or Microsoft XP
Internet Explorer	Obtain and install the appropriate language version of Internet Explorer 6.0 or higher prior to installing the G/Electric Adapter. See Note below.
RAM	256 MB minimum

Note: .NET Support Feature: The G/Electric Data Adapter is a .NET application that requires .NET Framework 1.1. If.NET Framework 1.1 is not installed on the personal computer; the Installer program will display the following warning message: "You can download the .NET Framework from Microsoft download site."

Disk Space Requirements

Installation Drive	5 MB
System Drive	Up to 1 MB Typically C:, where windows\system32 is located

Installing the G/Electric Data Adapter

The Oracle Utilities Network Management System G/Electric Data Adapter Installer automates the tasks that must be performed in order to properly configure and install the Oracle Utilities Network Management System G/Electric Data Adapter.

Obtaining the Software

The adapter setup files are included in the Oracle Utilities Network Management Optional Windows Applications download zip file that can be obtained from the Oracle e*Delivery site. After downloading the zip file, find the appropriate Adapter zip file within the NMS_GIS_Adapters directory and unzip to a location on your Windows server. Application Installation

Application Installation

If a previous version of the Adapter exists on the workstation, remove all the previous versions first. However, be sure to keep a backup copy of the application's configuration database file for use with the new software. Notices of schema changes for this database can be found in the readme.rtf file in the installation directory.

If this is the first installation, then an empty application configuration database file can be obtained from the installation package. Refer to the section labeled "Adapter Configuration Database" in this document for details on this required file.

Be certain to install the software while logged on to the workstation as a Windows Administrator.

Executing the Adapter Installer

Follow these steps to install the adapter:

1. After verifying that the Adapter installation personal computer meets the system requirements, copy the Oracle Utilities Network Management System G/Electric Data Adapter Installer supplied to you by Oracle to the desktop of the personal computer you will be running the installation on. It can be found in the NMS_V1.10.0_Intergraph_GIS_Adapter subdirectory.
2. Double-click the setup.exe file to start the installation. After a short time, the initial splash screen will display.
3. Follow the on-screen instructions and prompts as required to advance to the next screen. Accept the default values unless they conflict with your system's configuration (e.g., installation directory). The installation screens are listed below in the order they appear:

- Welcome Screen. This screen describes the installation process.
- Customer Information Screen. This screen prompts you for your name and your organization's name.
- License Information Screen. This screen displays the license agreement. You must click the I Agree button in order to proceed.
- G/Electric Data Adapter Information Screen. This screen displays version information for the adapter.
- Select Installation Folder Screen. This screen allows you to specify the installation directory. You can also specify whether the adapter should be available to anyone who uses this computer or only to you.
- Confirm Installation Screen. This screen allows you to confirm the information you've entered before installation begins. If you need to change anything, use the Back button. Otherwise, click Next to proceed.
- Installation Complete Screen. This screen indicates that the installation was successfully completed.

System Changes

The Adapter does not add, modify or remove any of the Windows system DLL in the <windows>\system32 directory.

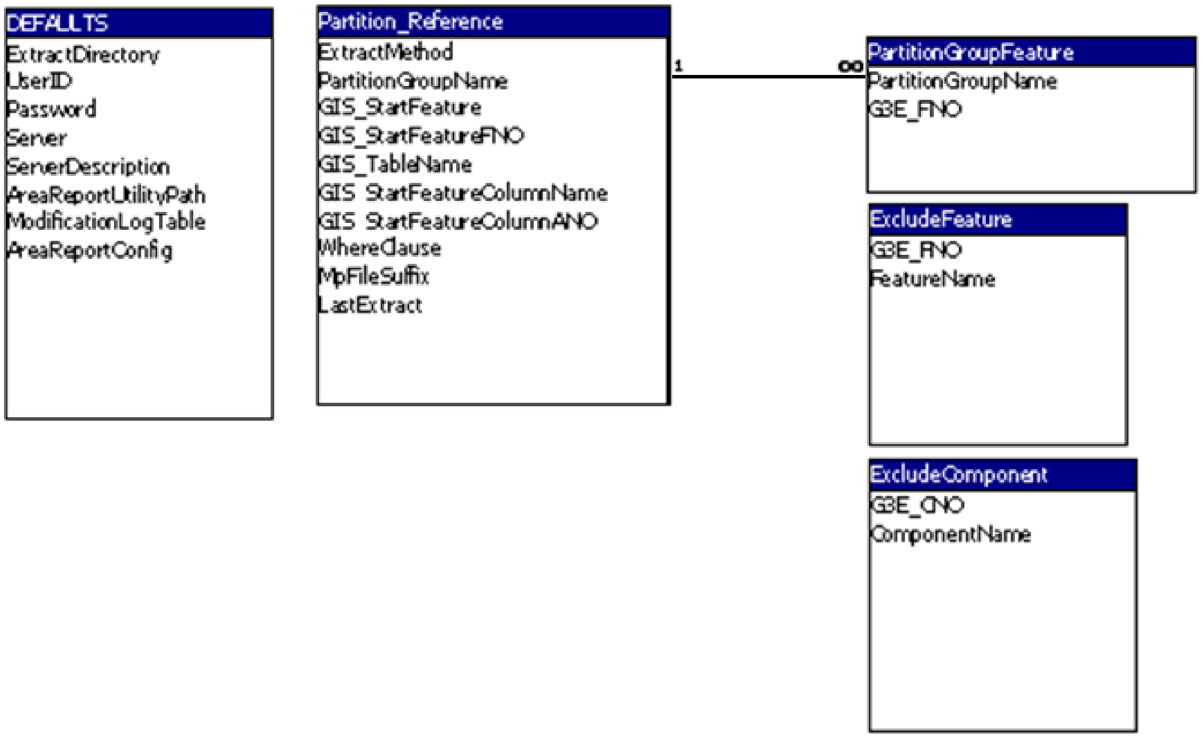
The Adapter installation program installs SmoothProgressBar.dll in the installation directory of the Adapter.

Following registry keys are created by the Adapter installation program:

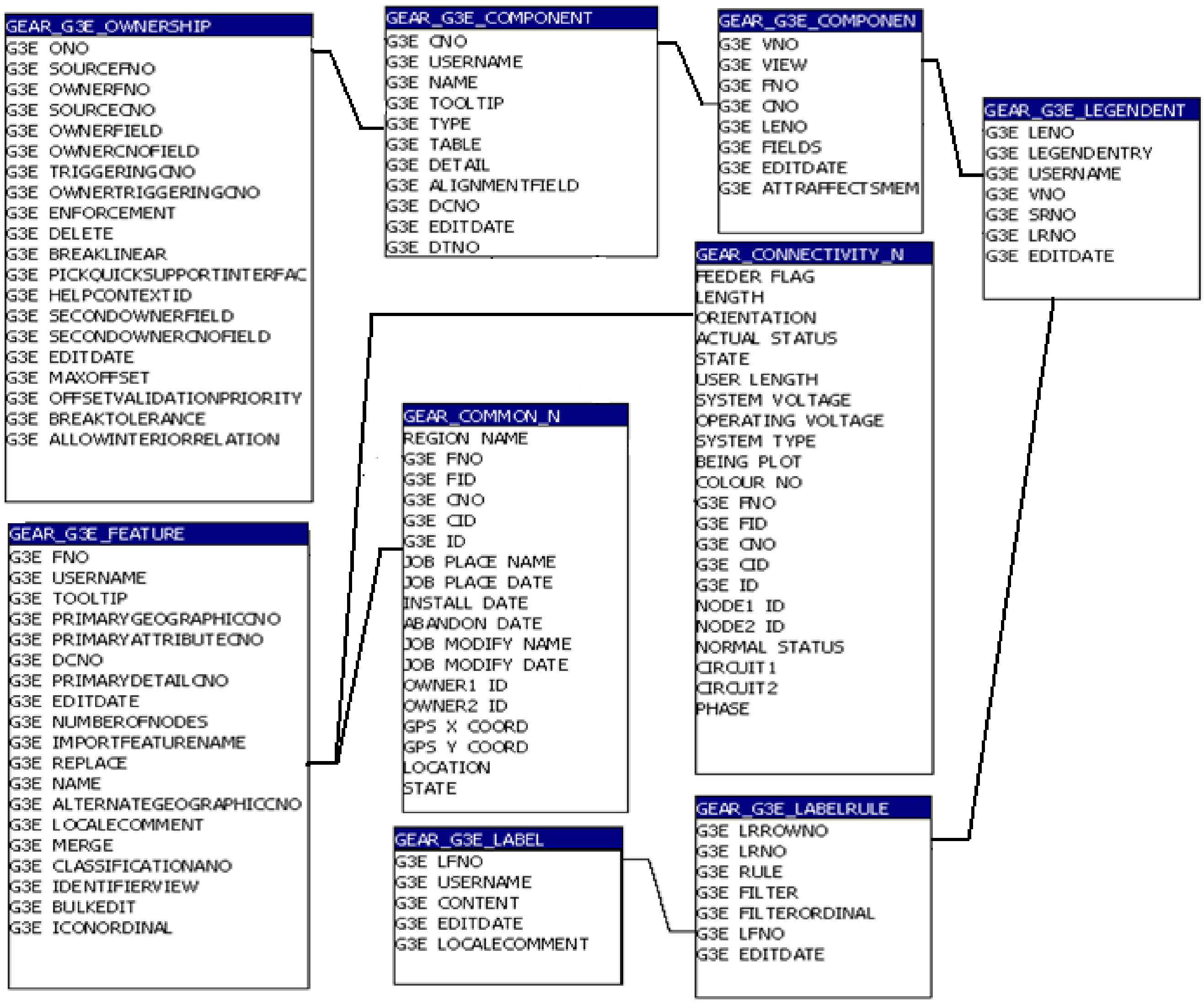
```
HKEY_LOCAL_MACHINE/Software/OracleTUGBU/Oracle Gelectric data
Extractor
String Value: VersionData: <Adapter version number>
```

Adapter Configuration

The Adapter is configured using tables in a Microsoft Access database, commonly referred to as the "application configuration database". The specific configuration tables, their contents, several examples, and content requirements are in the following Entity Relationship diagram.



The following diagram show the G/Electric Metadata Entity Relationship:



Defaults Table

The “Defaults” table is used to indicate the default settings for the Adapter. These settings should reflect the most common parameters used for an extract. The Adapter will read only the first row in this table to apply the default settings.

Column Name	Description	Type	Unique	Required?
ExtractDirectory	Directory for extracted data output	Text(100)	Yes	Yes
UserID	G/Electric database user id	Text(50)	Yes	Yes
Password	G/Electric database user password. (Masked in **** format)	Text(50)	Yes	Yes
Server	G/Electric database connection configuration string	Text(50)	Yes	Yes
ServerDescription	GIS Server information	Text(100)		No

Column Name	Description	Type	Unique	Required?
AreaReportUtilityPath	Directory of the AreaReportUtility.exe If the path is not provided or application is not found in the provide path then Adapter will disable the Spatial (landbase) extract.	Text(200)		No
ModificationLogTable	Name of the G/Electric table where the modifications log is replicated. This value is required only if a table other than the modification log must be queried to check the changed features.	Text(50)		No
AreaReportConfig	G/Technology configuration name from connection configuration map If the config path is not provided then AreaReportUtility will not execute. The Spatial (landbase) extraction will be disabled.	Text(50)		No
ChangeDetect_NonElectric_FNO	Non Electrical Features, that need to be identified & extracted during Change Detect Extraction process.	Text(200)		No
GetAnnotationCNO_AsAttributes	GIS annotation feature component number list (separated by comma), where annotations also need to be extracted as attributes.	Text(200)	No	No
IgnoreComponentGeometry_CNO	Allows grouping of all component geometries as one diagram. This can be helpful for linear features like conductors and bus bars. (LIST G3E_CNOs separated by comma).	Text(50)	No	No

Example

Column Name	Value
ExtractDirectory	C:\temp
UserID	Gear
Password	****
Server	GearGIS
ServerDescription	GEAR GIS Database
AreaReportUtilityPath	c:\program files\ips\gtechnology\program
ModificationLogTable	ModLog
AreaReportConfig	GEAR
ChangeDetect_NonElectric_FNO	208, 209
GetAnnotationCNO_AsAttributes	45
IgnoreComponentGeometry_CNO	301

Partition_Reference Table

The "PARTITION_REFERENCE" table is used to identify the partitions (electrical feeders or geographical tiles) to be used for extraction and their appropriate configurations. For example, an electrical feeder is a partition because all the devices that take part in forming a feeder are electrically connected. Similarly all land base features located within a certain polygon can form a land base partition due to their spatial relationship with the polygon (all the features are located WITHIN the polygon which shows a location-based spatial relationship). In essence, a group of features that are related to each other because of spatial or connectivity-based relationships form a partition.

The Adapter uses the PARTITION_REFERENCE data table to identify names of the available partitions in the G/Electric database.

Column Name	Description	Type	Unique	Required?
ExtractMethod	Data extract method to indicate whether the Adapter performs a spatial data extract (using polygon area) or a connectivity based circuit data extract (using electric circuit data) or a Detail window based data extract (using Detail window data)	Text(10)	No	Yes
PartitionGroupName	Unique Name of the partitions grouped together (e.g., electric feeder partitions or land base partitions). This is the primary key.	Text(50)	Yes	Yes
GIS_TableName	G/Electric Start Feature's attribute table name in the database; e.g., BREAKER_N or CONNECTIVITY_N or DETAIL_IND_LB	Text(50)	No	Yes
GIS_StartFeatureColumnName	Column name from the table specified in the above GIS_TableName parameter. The value from this column will be used to identify each unique partition and output MP files will be named using this value (if values are not unique then use the Where clause to filter out undesired rows) e.g., FEEDER_NO	Text(50)	No	Yes
GIS_StartFeature	G/Electric feature name that determines the partitions in a partition group (e.g., land base grid class name or head of the circuit for electric feeder data; e.g., breaker) This is not a required attribute if Non-Spatial (Electrical feeders) is provided from another table e.g., CONNECTIVITY. This is also not required for extracting Detail features.	Text(50)	No	No
GIS_StartFeatureFNO	G/Electric Feature Number of the GIS Start Feature (Only required when GIS_StartFeature is provided)	Number(5)	No	No
GIS_StartFeatureColumnANO	Unique name column G3E_ANO from G3E_ATTRIBUTES table	Text(50)	No	No

Column Name	Description	Type	Unique	Required?
GIS_WhereClause	Standard SQL Where clause to filter the number of partitions in the provided class, e.g., G3E_FID<10000	Text(100)	No	No
MPFileSuffix	MP file suffix to distinguish between different partition group MP files	Text(3)	No	No
LastExtractDate	The last successful data extraction date of the partition (populated by the Adapter). The user can provide From and To date to identify changed partitions. If the user does not populate “From Date” to determine changes, then this date is used as a “From Date” to define the time duration.	Date	No	No

Populate Feeder List Using a Source Feature

When an electric feeder in the G/Electric GIS originates at a particular GIS feature then this feature is considered as a “Source” feature. If all the feeder are associated with a particular “Source” feature (e.g., Breaker) then use following example to populate the Electrical Feeders list in the Adapter

Column Name	Value
ExtractMethod	NonSpatial
PartitionGroupName	Electric Feeder
GIS_TableName	BREAKER_N
GIS_StartFeatureColumnName	BREAKER_NUMBER
GIS_StartFeature	Breaker
GIS_StartFeatureFNO	300
GIS_StartFeatureColumnName	3000001
GIS_WhereClause	
MPFileSuffix	
LastExtractDate	

Populate Feeder List Using a Database Table

If all of your electrical feeders do not originate at a particular GIS feature (e.g., Breaker) OR source feature's attribute value does not populate the CIRCUIT1 and CIRCUIT2 attribute in the CONNECTIVITY table then feeder names should be retrieved using a database table where all feeder names are listed. For example you could use CONNECTIVITY table to identify all the available feeders in the system. In this case you do not need to populate GIS_StartFeature, GIS_StartFeatureFNO and GIS_StartFeatureColumnANO attribute values.

Column Name	Value
ExtractMethod	NonSpatial
PartitionGroupName	Electric_Feeder
GIS_TableName	CONNECTIVITY_N
GIS_StartFeatureColumnName	CIRCUIT_1
GIS_StartFeature	
GIS_StartFeatureFNO	
GIS_StartFeatureColumnANO	
GIS_WhereClause	SYSTEM_VOLTAGE='13.8'
MPFileSuffix	
LastExtractDate	15 Jun 2005

Populate Detail Window Name Using a database Table

All features within a Detail Window is extracted using the DETAIL_USERNAME attribute within the DETAIL_IND_LB table

Column Name	Value
ExtractMethod	Detail
PartitionGroupName	Detail
GIS_TableName	DETAIL_IND_LB
GIS_StartFeatureColumnName	DETAIL_USERNAME
GIS_StartFeature	
GIS_StartFeatureFNO	
GIS_StartFeatureColumnANO	
GIS_WhereClause	
MPFileSuffix	_dt
LastExtractDate	

Populate Spatial (Landbase) Partition List

The Adapter extracts all the landbase and non-electrical data spatially, i.e. using a polygon grid (or tile). You need to provide the Polygon grid feature information in the PartitionReference table to extract the non-electrical data.

Column Name	Value
ExtractMethod	Spatial
PartitionGroupName	Landbase
GIS_TableName	OMSGRID_N
GIS_StartFeatureColumnName	GRID_NUMBER
GIS_StartFeature	OMS_GRID
GIS_StartFeatureFNO	204
GIS_StartFeatureColumnANO	2040001
GIS_WhereClause	GRID_NUMBER<150
MPFileSuffix	_lb
LastExtractDate	

ExcludeFeature Table

The ExcludeFeature" table is used to indicate the G/Electric GIS features that should NOT be extracted by the Adapter.

Column Name	Description	Type	Unique	Required?
G3E_FNO	Unique and valid G/Electric G3E_FNO from the G3E_FEATURE table. This is the primary key.	Integer	Yes	Yes
FeatureName	Feature User Name (Not used by the Adapter. This is for the administrator's reference.)	Text(80)		

Example:

Column Name	Value
G3E_FNO	229
FeatureName	Region

ExcludeComponent Table

The ExcludeComponent" table is used to indicate the G/Electric GIS components that should NOT be extracted by the Adapter. If a feature is listed in the ExcludeFeature table, then listing individual feature components in the ExcludeComponent Table is not required because all the components of the features will be excluded anyway. If a global component (such as Leader Line) needs to be excluded from the data extract, then it should be listed here only once.

Column Name	Description	Type	Unique	Required?
G3E_CNO	Unique and valid G/Electric G3E_CNO from the G3E_COMPONENT table. This is the primary key.	Integer	Yes	Yes
ComponentName	Component User Name (Not used by the Adapter. This is for administrator's reference)	Text(80)		

Example:

Column Name	Description
G3E_CNO	33
ComponentName	Hyperlink Attributes

PartitionGroupFeature Table

The "PartitionGroupFeature" table is used to specify the spatial "Partition Group Name" to G3E feature relationship. The user can define the spatial PartitionGroupName in the Partition_Reference table and specify which G/Electric features belong to the partition group.

The user can extract grouped GIS features in separate MP files based on the PartitionGroupName. For example, the spatial PartitionGroup named "Street" will be extracted in a separate MP file where features listed in the PartitionGroupFeature table will be extracted.

Column Name	Description	Type	Unique	Required?
PartitionGroupName	Unique Name of the partitions grouped together (e.g., electric feeder partitions or land base partitions) Populated from PartitionReference table	Integer	Yes	Yes
G3E_FNO	GIS Feature Number of a feature which should be extracted as a part of the PartitionGroup. This is the primary key.	Text(80)		

Example:

Column Name	Description
PartitionGroupName	Street
G3E_FNO	182

V_SPL_FIDFNO View

The G/Electric Adapter requires an Oracle view called V_SPL_FIDFNO in the G/Electric Oracle Database. The system administrator must create the view V_SPL_FIDFNO on the G/ Electrical Oracle database server.

The V_SPL_FIDFNO view provides the G3E_FNO (feature number) for every G3E_FID in the G/Electric database. When the Adapter executes the G/Electric's spatial query application AreaReportUtility to find the features within the specified spatial area, the application returns only the G3E_FID of the features. The V_SPL_FIDFNO view provides the feature number (G3E_FNO) of every G3E_FID (unique feature identifier) in the system, which reduces the overhead of querying each G3E_FID to find the corresponding G3E_FNO.

The system administrator should execute the create_spl_view.sql file on the G/Electric GIS server using the SQL PLUS utility. The create_spl_view.sql file is located in the Reference subdirectory of the Adapter installation directory.

The system administrator should edit the create_spl_view.sql file to make sure that the GIS COMMON component number is correct in the file. The create_spl_view.sql file lists "G3E_CNO=31". If the COMMON component number (G3E_CNO) in your G/Electric GIS database is different, then the G3E_CNO must be replaced with the correct G3E_CNO of the COMMON component

The query result returned by the database should be captured in a text file by using the SQL Plus spool command.

The spooled file should be edited to complete the Create view definition by:

- In the beginning of a spooled file add a statement:
  - Create or Replace View V_SPL_FID_FNO as
- After the last union all statement in the spooled file add a statement:
  - Select G3E_FID, G3E_FNO from COMMON_N;
- Execute the spooled file using the Oracle SQL Plus application to create the V_SPL_FIDFNO view.

If the V_SPL_FIDFNO view is not available, then the Adapter will disable the spatial (land base) data extract functionality and only non-spatial extraction (electric feeders) can be carried out.

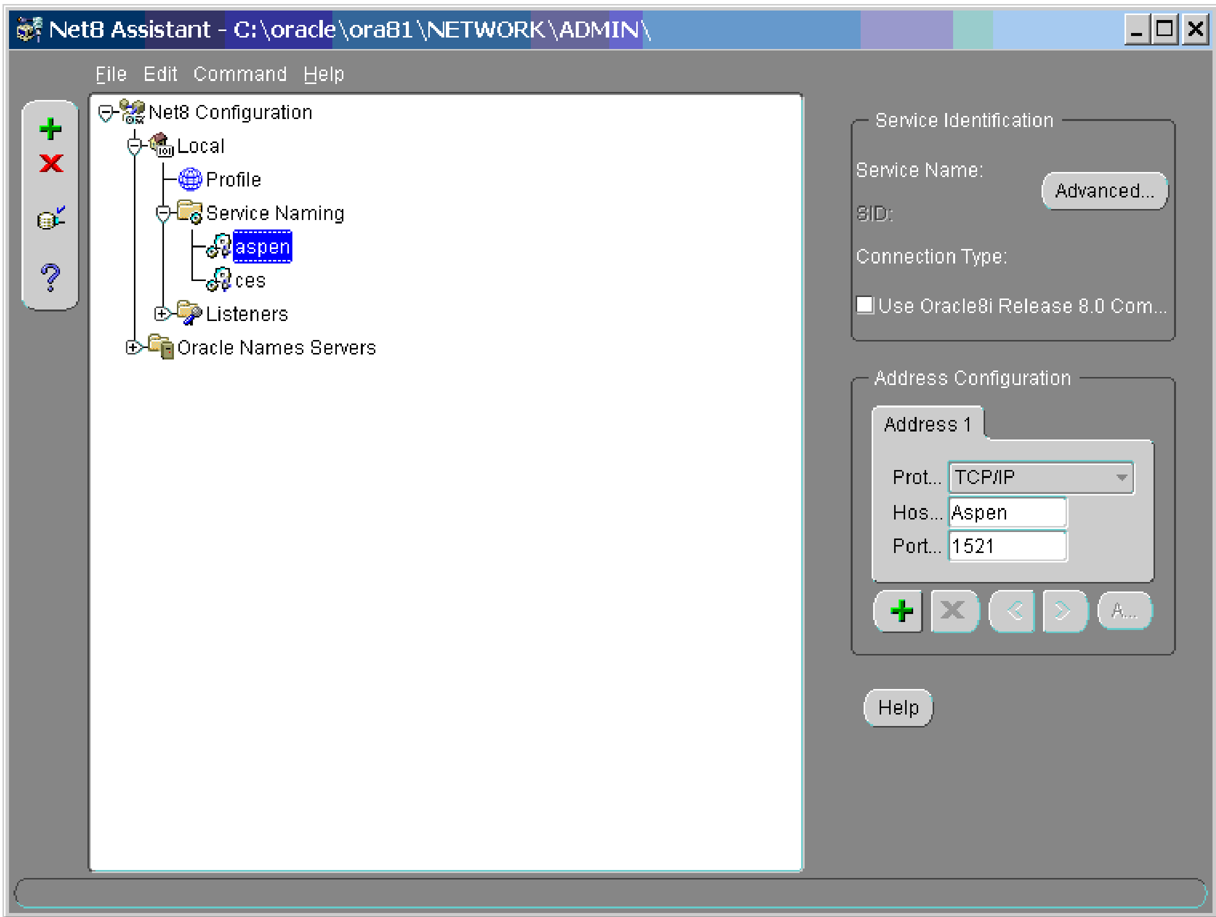
Third Party Products

The G/Electric Data Adapter requires three third party products. Those products are:

- G/Electric GIS Database Server
- Microsoft Access to configure the Adapter configuration database
- Oracle SQL*NET Client

Oracle Client Configuration

The Adapter requires the Oracle client installed on the machine where the Adapter is installed. The Adapter administrator must configure the Oracle server name using the Oracle Net Configuration Assistant utility. The Oracle server name must be populated in the "Server" column of the Adapter configuration database's "Default" table.



Executing the Adapter

The Adapter can be run either in the background with command line parameters or interactively with a GUI. In any event, ensure the Adapter configuration database file GElec2Oracle.mdb has been customized prior to running the Adapter for the first time.

When the Adapter is launched interactively or in command line mode, it creates the "initialize.log" file in the Adapter installation directory. The log file reports initialization status of the Adapter. Once the Adapter is launched successfully and extraction has started, the "extracting.log" file is created in the Extract Work Order directory.

Launching the Adapter Interactively

The G/Electric Data Adapter can be launched from the Windows Start button, Programs, Oracle's G/Electric Data Adapter. After a few moments, the G/Electric Data Extractor configuration window is displayed.

This window contains four main framed panels:

- Change Detection
- Data Extraction Parameters
- Status panel – Where information on the extraction process is reported to the user.
- Work Order Details and the Launch Extract and Cancel buttons.

Debug Mode

The Adapter could be launched with debug mode on to report more detailed reporting of the Adapter activity in initialize.log and extracting.log files. In the debug mode the Adapter writes every detail in the log file, so it is very time consuming. The Adapter should be launched in the debug mode only during the test phase or to identify cause of the Adapter error.

To run the Adapter in debug mode:

- Create a new shortcut icon on the desktop of the computer for launching the Adapter software (GElec2Oracle.exe)
    - In 'Create Shortcut' dialog, browse to the Adapter installation directory and select GElec2Oracle.exe. Click Next button to display the 'Select a Title for the Program' dialog.
    - After providing the name for the short cut click Finish button to create the short cut.
- Right mouse click on the newly created shortcut on the desktop and select Properties from the displayed menu. The system will display Properties dialog with Shortcut tab selected
    - In the Target Section for the shortcut, specify the flag –DEBUG at the end of the existing value. For example
        - Before: C:\Program Files\OracleTUGBU\Oracle GElectric Data Extractor \GElec2Oracle.exe
        - After: "C:\Program Files\OracleTUGBU\Oracle GElectric Data Extractor \GElec2Oracle.exe" –DEBUG (You need to provide double quotes enclosing the Adapter application path but make sure –DEBUG is not typed within the closing double quotes.
    - Click Apply button to dismiss the Properties dialog.
- Launch the Adapter using newly created shortcut. The log files will provide a lot more information.

Data Extraction Parameters Panel

This panel contains the data parameters required for conducting the extraction. In order to extract data partitions, the user needs to select a partition group name and partitions that are part of the desired partition group.

The partition group names are populated using the PartitionGroupName column of the PARTITION_REFERENCE table of the Adapter configuration database. The Adapter finds all the partitions belonging to the defined group by querying the G/Electric database.

Depending on the partition group name chosen by the user, the items (partition names) listed in the "Partitions" list box on the right hand side of the panel will change to reflect the unique partition names associated with the specified partition group.

The user can manually choose which partitions to extract from the partitions list on the right hand side of the GUI. Note that clicking or highlighting a partition on the partitions list will queue it for extraction. Standard Windows Shift-click or Alt-click functionality is employed for multiple selects.

Other tools include the Select All and Unselect All buttons, which will select all the partitions in the queue, or none of them respectively.

Extract Launch Controls

The Adapter displays the extracted data directory path where work order sub-directories will be created. The extract directory path is read from the ExtractDirectory field of the “Defaults” table in the Adapter configuration database.

The extracted data will be extracted in a work order directory created under the default extract directory. The default name for an extract directory is a combination of the system’s current date and time (e.g., 2001Aug30-1256); however this can be overridden by simply changing the contents of the Work Order Name field.

Pressing the Launch Extract button begins the extraction of the selected partitions. Once the extract begins, the Exit button in this panel is replaced with a Cancel button, which - if clicked - will allow the user to interrupt the extract process.

Change Detection Panel

This panel contains information for the change detection functionality of the Adapter. It also displays the G/Electric GIS server information and date of the last data extract for the selected partition group name.

The change detect mechanism identifies the changed partitions over a period of time. The user needs to provide a time range using the “From Date” and “To Date” fields. By default, the Adapter enters the Last_Extract date in the “From Date” and the current Adapter launch date and time in the “To Date” field, but the user can override these default values. If no data extraction has taken place, then the From and To date fields will be populated with the current Adapter launch date and time.

If the user has selected a “Non Spatial” partition group (i.e., electric), then the system will detect the changed feeders between the defined time duration. Similarly, if a user has selects a “Spatial” partition type (e.g., land base), the system will then detect changed land base polygons (partitions) between the defined time duration.

G/Technology provides GIS work order management through the concept of a long-term transaction (LTT), commonly referred to as a Job. It allows GIS users to perform Long Term Transaction (LTT) changes (unapproved) in the database while keeping these changes isolated from other LTTs (users working on the approved, master records and those working on their own, as of yet unapproved, LTTs). These changes can be deletions of and modifications to existing records as well as adding additional records. In this way GIS users can initiate an LTT and perform inserts, updates, and deletes in that LTT.

When changes are complete, the GIS user posts them to the master database by running the “Post Job” command. This ends their isolation from the master and from other GIS users. When this happens, the GIS user's changes become part of the master record, replacing the current master record. The GIS user must be editing the job in order to post it. Once the job is posted, the feature’s LTT_STATUS is set to null.

The Adapter will extract only master database records with the LTT_STATUS of NULL or ‘COPIED” and the change detect routine will use only master database records to identify the changes. The Adapter will not extract any of the LTT data or consider the changes made by the LTT in order to identify the changed partitions.

The Adapter uses the G/Electric facility model’s ModificationLog table to identify the changed or deleted features. The Adapter will use the user-provided time range to identify changed or deleted features, but considers only those features that are specified in PartitionGroupFeature table for the selected partition group name. Based on the changed feature's location or connectivity, the Adapter will determine the changed partitions. For example, if the partition group is a spatial partition group then a changed polygon will be reported OR if the partition group is a non-spatial (electric) partition, then a changed feeder will be reported.

Executing the "Perform Change Detect" command will automatically queue partitions that contain changed features between the user provided "From" and "To" dates. The change detection process requires intensive database searches and comparisons to identify changed partitions.

To perform Change Detect of Deleted Features, GElectric Adapter, creates a master reference table CONNECTIVITY_SPL, to store all the G3E_FID, CIRCUIT1 & CIRCUIT2, LTT_DATE of all electrical features, in GIS database. The Adaptor uses the CONNECTIVITY_SPL & the modification log table, to identify any deleted features.

Modification Log

If departmental policy or business processes require the GIS administrator to purge the ModificationLog table frequently, quick and easy access to the G/Electric feature modification information is lost. The G/Electric database provides alternate ways to find the added and modified features but deleted feature information will be lost.

If the extract frequency is going to be longer than the ModificationLog table purge frequency, then it is advisable to copy the ModificationLog table in an intermediate table for the Adapter. The Adapter will use the modification log data from this new Adapter-specific table to identify the changed partitions. The table name must be specified in the ModificationLogtable column of the Default table in the Adapter configuration database.

Status Panel

Because the extraction process will take time to complete, its status and progress can be monitored from the panel at the bottom of the GUI. The Overall Progress field displays an estimated extraction progress of all the queued partitions, while the Detail Progress field indicates more detailed extraction progress information, such as the specific partition and feature class currently being extracted.

Launching the Adapter in Batch Mode

The Adapter can be launched in a non-GUI mode from the Windows command line with parameters. However, because this launch method is "silent," no visual feedback will be provided to the workstation screen except for error reports. The user will have to view the output directory in order to determine progress.

The Windows service scheduler can be used to schedule an automatic extract process.

When launching the Adapter in non-GUI mode, only the non-GUI launch flag is required, while all other command line parameters are optional. If a parameter is omitted, the default is assumed. In default mode, the Adapter extracts the electric feeder data.

The command line usage is:

```
"[install directory]\ GElec2Oracle.exe" -NONGUI [-DEBUG] [-?] [-d
dataset] [-r reference] [-p partition] [-w workorder] [-o] [-q queue]
```

Where:

-NONGUI	(required) Indicates to launch Adapter in non-GUI mode.
-DEBUG	Launches the Adapter with Debug mode on.

-?	Command Line Help
-r reference	Path of application reference database file if other than the default GElec2Oracle.mdb in the extract install directory (e.g., -r c:\myrefdb.mdb)
-p partition	Partition type to use; as in PartitionReference table (e.g., -p Electric_Tile)
-w workorder	Workorder directory to create (e.g., -w Nov20-01)
-o	Force an Overwrite of existing workorder and version (if existing ones are found)
-q queue	Method to queue partitions: ALL - queues all partitions (e.g., -q ALL) CD - uses change detection to queue (e.g., -q CD) (default) LIST - user provides a list of partition names to be extracted which are separated by a comma. (If multiple partition types are in provided using –p option where Adapter cannot determine the PartitionGroup of listed partition queue then Adapter will report the warning and terminates.

Output of Adapter

The model preprocessor import files (.MP), the log file, and the work order time stamp file are all placed in a folder with the work order name. This folder will appear in the output directory as designated in the “Defaults” table. If change detection was run, a log file of the process will be placed in the root of the output directory.

Note that if the extract is still in progress or does not complete, the work order time stamp file will not appear in the output directory. The work order time stamp file will only appear if the complete extract finishes successfully. As such, external processes can use its presence as an indicator of a completed extract.

Performance

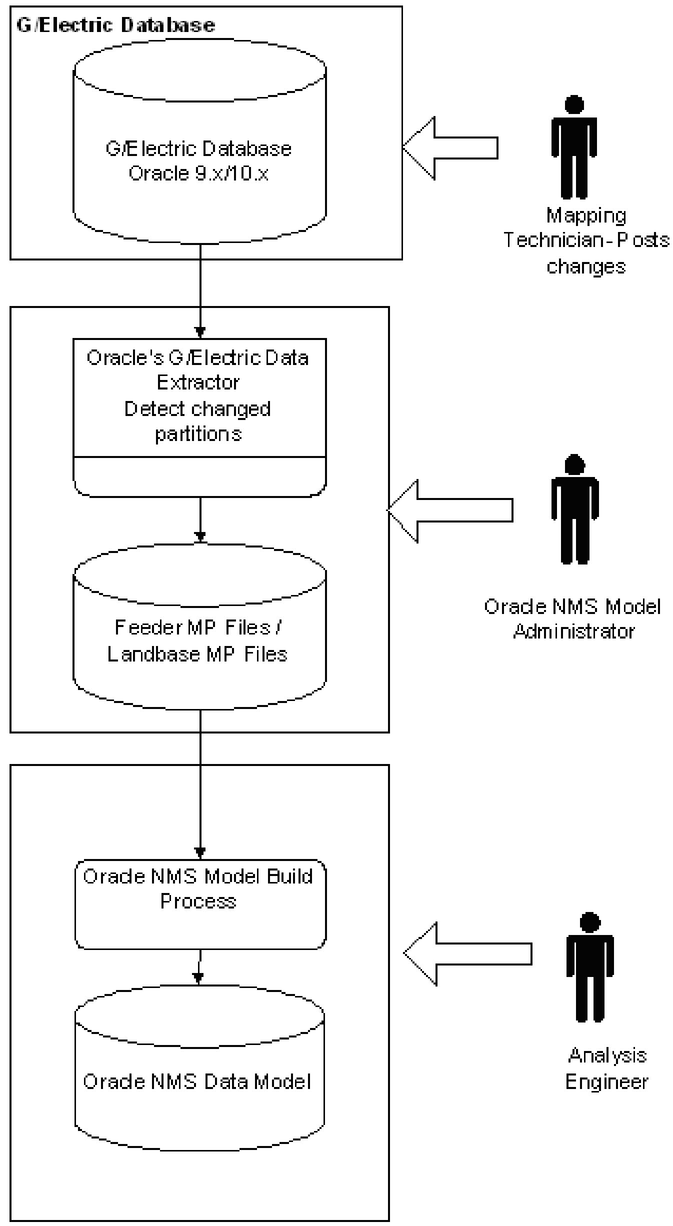
The non-spatial data extract (i.e., of electric feeders) does not depend upon the GIS’ AreaReportUtility application to find the features located within a polygon partition so these extracts are faster compared with a spatial data extract. The spatial data extract uses G/Electric’s AreaReportUtility application to identify features located within a polygon area.

The number of class components being extracted will dramatically affect the Adapter performance. Therefore, if any of the features components are not required for the Oracle Utilities Network Management System then they should be listed in the ExcludeComponent table. The ExcludeComponent table ensures that feature information will be extracted but components listed in the ExcludeComponent table will not get extracted. For example detailed document, raster, or hyperlink document components are not required for the Oracle Utilities Network Management System. These components should be listed in the ExcludeComponent table.

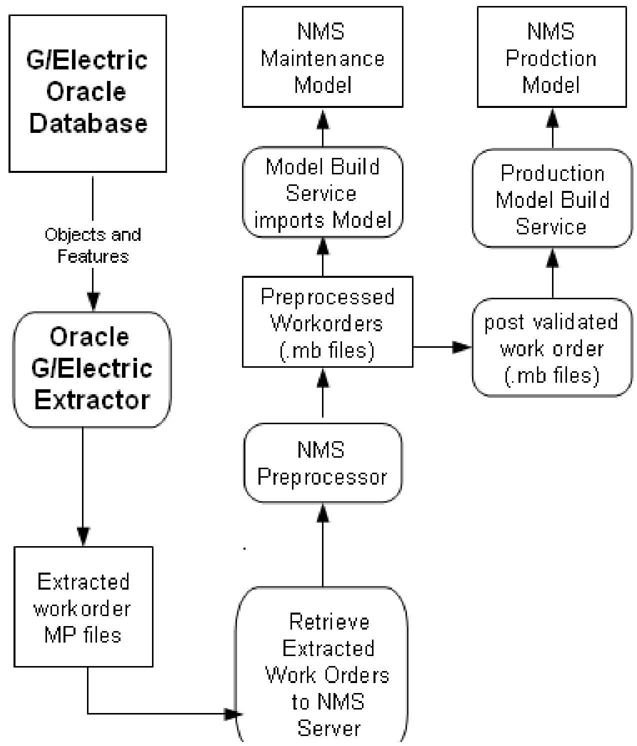
The time required to perform change detection will vary and is dependent on the number of features that have changed since the last extract. The list of changed features does not provide the partition information, so the Adapter depends upon G/Electric’s AreaReportUtility to identify the spatial polygon (partition) of the changed feature.

Application Process Flows

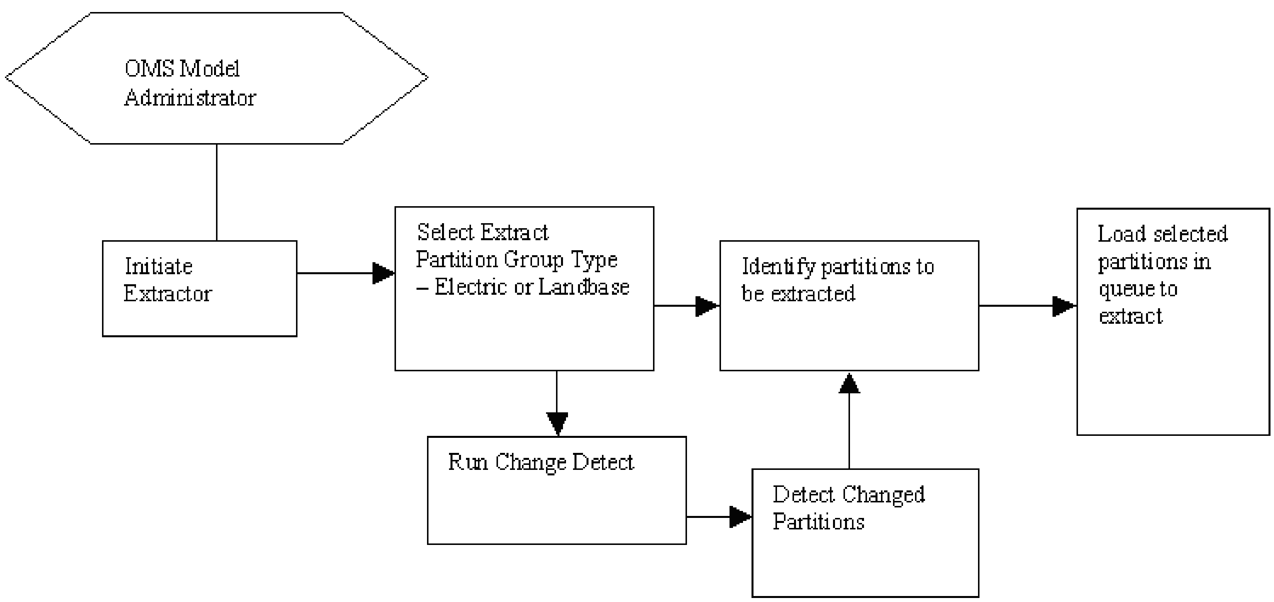
The following figure shows the G/Electric Adapter user process flow:



The following figure shows the G/Electric Adapter module process flow:



The following figure shows the G/Electric Adapter process flow:



GIS Data Requirements

Required Data for All Adapter Features

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Object type	G3E_FNO and G3E_USERNAME, Feature Class and attributes	
Object Unique ID	G3E_FID	
Object geometry (point, line, text)	Feature Geometry	Adapter produces diagram object of type point, line, or text.
Points: (x, y) location angle, justification	Point Feature Geometry	Adapter produces diagram object characteristics of: location, angle, and scale.
Lines: List of coordinates	Linear Feature Geometry	Adapter produces diagram object characteristics of: location, angle, scale
Text: (x,y) location angle or endpoint string justification	Feature Text Component	Adapter produces diagram object characteristics of: location, angle, string, and justification.

Optional Data for All Adapter Features

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Diagram declutter ranking value	N/A	Not retrieved
(x,y) offset for graphical attributes	N/A	Not retrieved
Diagram groupings for multi-object hide/display	N/A	Not retrieved
Symbology class	N/A	Not retrieved
Scale factor	Attribute	E.g., urban or rural attribute prescribes a certain scale factor

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Object location ID	Based on Partition	

Required Attributes for Electrical Objects

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Connectivity information	NODE1 and NODE2 from CONNECTIVITY component	
Normal status of device	NORMAL STATUS from CONNECTIVITY component	
Orientation of the feature – OH or UG	ORIENTATION from the CONNECTIVITY component	
Phases for the device (PHASES)	PHASE from CONNECTIVITY component	
Voltage level (VOLTS)	Voltage from CONNECTIVITY component	

Optional Attributes for Electrical Objects

Oracle Utilities Network Management System Requirement	GIS Source	Comments
Network protection direction	N/A	
Operations control authority zone (NCG)	N/A	

Sample Model Preprocessor File

The main products of the Adapter are the model preprocessor (".mp") files. There is one preprocessor file for each partition.

A sample of a model preprocessor file that contains a partition identifier (feeder 1) and Switch feature (with switch unit and text annotation) is shown below:

```
ADD Feeder1 {
NAME = Elec1;
COORD_SYSTEM = NAD_1927_StatePlane_Pennsylvania_South_FIPS_3702;
SCALE_FACTOR = 1.0;
Diagram[Feeder] = {

    GEOMETRY ={
```

```
                            (-2147000000.00000,-2147000000.00000),
                            (2147000000.00000,-2147000000.00000),
                            (2147000000.00000,2147000000.00000),
                            (-2147000000.00000,2147000000.00000),
                            (-2147000000.00000,-2147000000.00000);
        };
    };

ADD Primary_Switch 815425 {
   PORT_PORT_A=129493717;
   PORT_PORT_B=129493718;
   DIAGRAM[31303] = {
    HEIGHT = 0;
    ANGLE = 0.3130386;
    SCALE = 0;
   TEXT = "445SWITCH_NUMBER";
   GEOMETRY = {
   (-553069.9375,-20248.82421875)};
};
   DIAGRAM[31302] = {
    HEIGHT = 0;
    ANGLE = 1.747896;
    SCALE = 0;
   GEOMETRY = {
   (-553076.625,-20250.107421875)};
};
   ATTRIBUTE[G3E_FID]="815425";
   ATTRIBUTE[G3E_FNO]="313";
   ATTRIBUTE[FEATURENAME]="Primary Switch";
   ATTRIBUTE[SWITCH_NUMBER]="445";
   ATTRIBUTE[RATING]="600.00";
   ATTRIBUTE[SWITCH_OPERATOR]="MANUAL";
   ATTRIBUTE[SWITCH_TYPE]="AIR BREAK";
   ATTRIBUTE[SWITCH_SUB_TYPE]="Load Break";
   ATTRIBUTE[LTT_ID]="0";
   ATTRIBUTE[MIRROR]="NO";
   ATTRIBUTE[JOB_PLACE_NAME]="MetadataEdit";
   ATTRIBUTE[JOB_PLACE_DATE]="05/14/2004 1:34:57 PM";
   ATTRIBUTE[JOB_MODIFY_NAME]="JC_May20";
   ATTRIBUTE[JOB_MODIFY_DATE]="06/01/2004 2:30:12 PM";
   ATTRIBUTE[STATE]="In Service";
   ATTRIBUTE[ASSET_OWNER]="THESL";
   ATTRIBUTE[IPID]="35503";
   ATTRIBUTE[DISTRICT]="YK";
   ATTRIBUTE[LTT_ID]="0";
   ATTRIBUTE[LTT_DATE]="06/01/2004 2:30:12 PM";
   ATTRIBUTE[LTT_TID]="392702";
   ATTRIBUTE[REGION_NAME]="York";
   ATTRIBUTE[NORMAL_STATUS]="CLOSED";
   ATTRIBUTE[CIRCUIT1]="11-M1";
   ATTRIBUTE[CIRCUIT2]="11-M1";
   ATTRIBUTE[PHASE]="RWB";
   ATTRIBUTE[ORIENTATION]="OH";
   ATTRIBUTE[ACTUAL_STATUS]="CLOSED";
   ATTRIBUTE[STATE]="In Service";
   ATTRIBUTE[SYSTEM_VOLTAGE]="27.6";
   ATTRIBUTE[OPERATING_VOLTAGE]="27.6/16 kV";
   ATTRIBUTE[COLOUR_NO]="1";
   ATTRIBUTE[LTT_ID]="0";
   ATTRIBUTE[FEEDER_FLAG]="NO";
};
```

Uninstalling the G/Electric Data Adapter

Microsoft Windows 2000

From the Start button, click Settings, then Control Panel. Double-click the Add/Remove Programs Icon. Select the G/Electric Data Adapter application from the program list, and click the Change/Remove button.

Microsoft Windows XP

From the Start button, click Control Panel. Double-click the Add/Remove Programs Icon. Select the G/Electric Data Adapter application from the program list, and click the Remove button.

Chapter 6

Generic WebSphere MQ Adapter

This chapter includes the following topics:

- **Introduction**
- **Hardware and Software Requirements**
- **Functional Description**
- **High Availability**
- **Performance**
- **Data Flows**
- **Information Model**
- **Configure Queues for Required Data Flows**

Introduction

The purpose of this document is to provide an administration guide of the Oracle Utilities Network Management System Generic WebSphere MQ Adapter. This document will discuss the required process for installing and configuring the adapter to run with the appropriate Oracle Utilities Network Management System software.

The Oracle Utilities Network Management System Generic WebSphere MQ Adapter can serve as a data adapter between the Network Management System and a number of external systems, such as a Customer Information System (CIS) or an Interactive Voice Response (IVR) system. Additionally, this adapter could be used with external systems that need to input trouble calls into Network Management System but do not require other sophisticated data flows, such as High Volume Call Applications (HVCA), Automated Meter Reading (AMR), or Work Management Systems (WMS).

This interface depends upon IBM's WebSphere MQ software as the intermediary repository for information passed between the Network Management System and the external system. This adapter is used only for the communication between IBM's WebSphere MQ software and the Oracle Utilities Network Management System software suite. With the use of XML as the payload for data transmission, the adapter exchanges data between Oracle Utilities Network Management System and external systems as defined in the Oracle Utilities Network Management System WebSphere MQ XML schema documents. The required configuration of the adapter is described in the sections that follow.

Terminology

The following terms and acronyms are relevant to this specification

OMS	Outage Management System
Network Management System	Network Management System
SMService	System Monitor Service. SMService monitors the core processes in the system, essentially the services and interfaces.
JMService	Job Management Service. The Oracle Utilities Network Management System call processing and outage prediction engine.
MQSeries	A queue-based messaging system developed by IBM. This system has been renamed to WebSphere MQ.
DTD	Document type definition, used to define XML documents
XML	Extensible Markup Language
XSL	XML Style Sheet, used to reformat XML documents
XML Schema	An XML standard for defining XML documents
CIS	Customer Information System
IVR	Interactive Voice Response system
SCADA	Supervisory Control and Data Acquisition system
HA	High availability, where Oracle Utilities Network Management System is configured with a pair of redundant servers. This is usually in the form of a hardware cluster and a shared drive that contains the database.

Hardware and Software Requirements

Oracle Utilities Network Management System Environment

The Oracle Utilities Network Management System environment consists of a number of servers that are interconnected using the InterSys messaging system.

Adapter Server

The Generic WebSphere MQ Adapter environment may be resident on the same servers as the Oracle Utilities Network Management System services, or it may be implemented on a separate server. Specifications for a stand-alone adapter server:

- All Oracle Utilities Network Management System UNIX operating systems are supported.
- IBM WebSphere MQ messaging product must be installed. Note, however, that the queues may reside on a remote machine.
- A LAN connection to the Oracle Utilities Network Management System server must be available.

- ISIS must be installed and configured

Depending upon the high availability scheme selected, it would be possible to configure more than one adapter server for redundancy.

Oracle Utilities Network Management System Server

The Oracle Utilities Network Management System server environment is typically deployed on one or more UNIX servers configured with the following:

- UNIX operating system
- Oracle RDBMS with Oracle Utilities Network Management System model
- Oracle Utilities Network Management System service processes
- LAN connection to adapter server
- Message queues to be used by the MQ/XML Adapter appropriately declared in the defined database configuration table.
- ISIS

External System Environment

The external system is any system that can exchange information with Oracle Utilities Network Management System through an adapter. The environment of the external system has the following capabilities:

- Any operating system which supports IBM WebSphere MQ messaging
- IBM WebSphere MQ messaging product
- Applications that can request or publish information in a manner which is either directly or indirectly (through a translator) compliant with the XML specifications contained within this document via queues
- Queues must be pre-configured
- IBM WebSphere MQ Integrator can be used as needed for routing and translation.

Required Installed Software:

The following lists the required software that needs to be installed prior to any configuration of the Oracle Utilities Network Management System Generic WebSphere MQ Adapter.

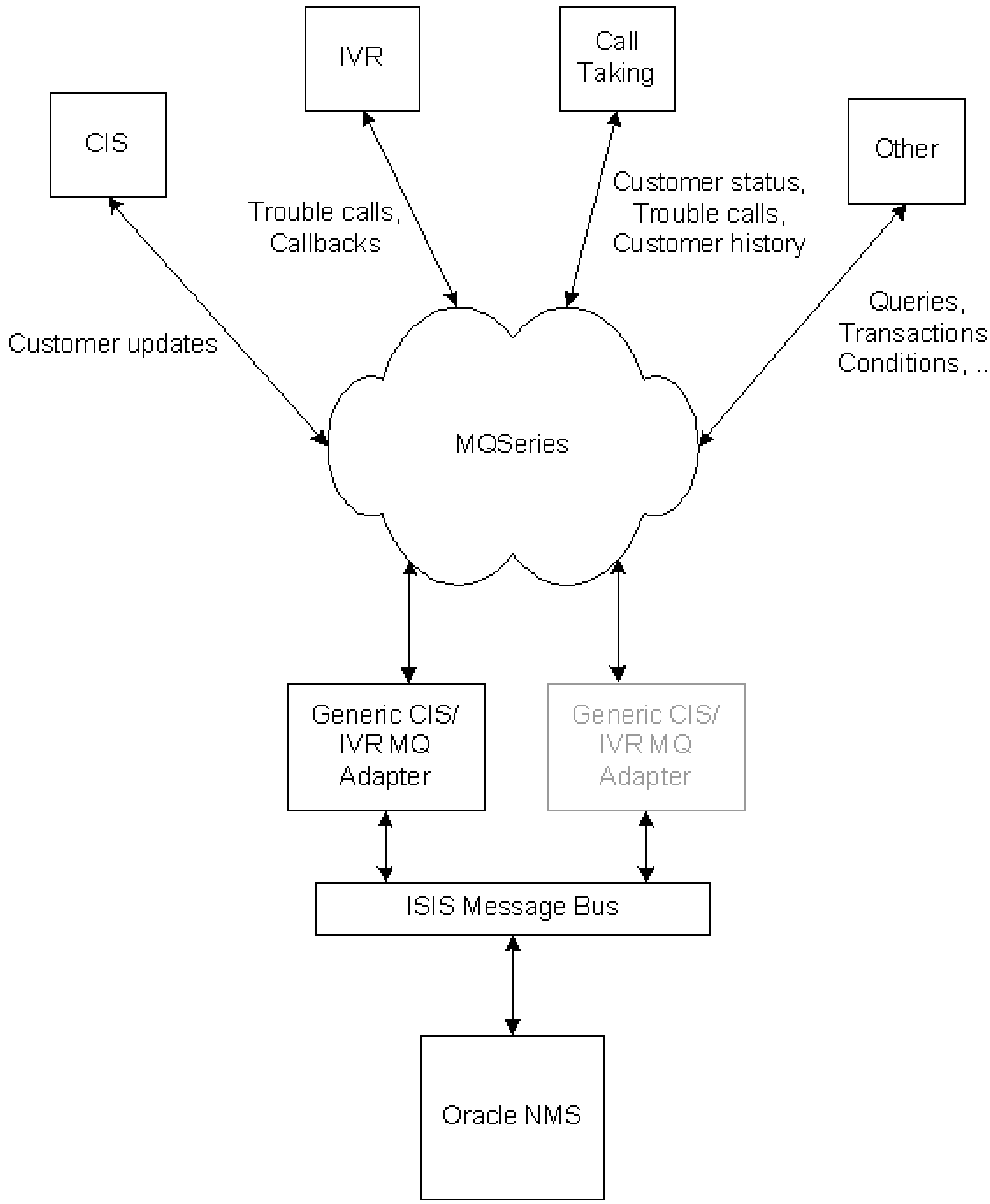
- IBM’s WebSphere MQ
- ISIS (installed as part of the base Oracle Utilities Network Management System installation)

  Note: ISIS is the messaging backbone for Oracle Utilities Network Management System and will already be present on any Network Management System servers. If the Generic WebSphere MQ Adapter is to be executed on a separate server than the Network Management System, then that server must also have ISIS installed and running. Every server installation must be running the same version of ISIS. The CMM_CELL environment variable must be set the same on any servers which are to communicate through ISIS.

Functional Description

Context Diagram

Below is a diagram of the interaction between Oracle Utilities Network Management System and various external applications via the Generic WebSphere MQ Adapter.



WebSphere MQ Gateway Context

In this document, it is assumed that the Generic WebSphere MQ Adapter's tables reside in the database used by Oracle Utilities Network Management System.

Adapter Installation

Overview

This section guides the user in the installation of the Oracle Utilities Network Management System Generic WebSphere MQ Adapter. The following are assumed to be true before the adapter is installed:

1. Oracle Utilities Network Management System is installed and functional. This means that database access has been confirmed, as well as ISIS message bus communication.
2. WebSphere MQ is installed on a machine that is accessible to the Oracle Utilities Network Management System.

Check if the Generic WebSphere MQ Adapter is installed

Verify that the following files are found in their respective folders

- $CES_HOME/bin/mqseriesgateway
- $CES_HOME/bin/ces_mq_gateway.ces

Configure Adapter to run as NMS System Service

Configure the Generic WebSphere MQ Adapter to run as an Oracle Utilities Network Management System service by updating the `$NMS_HOME/etc/system.dat` file to include the Generic WebSphere MQ Adapter as a system service. There are 3 main sections where this service needs to be defined: the service, program and instance sections.

See the `$CES_HOME/templates/system.dat.template` file for examples of how to configure the Generic WebSphere MQ Adapter. Search for "`mqseriesgateway`" in the file and copy those lines to the `$NMS_HOME/etc/system.dat` file. Make sure all lines are uncommented so that they are active. See the command line options section below for more details on available options. You must restart the system services in order to the Generic WebSphere MQ Adapter to properly be monitored by SMService.

> **Note:** In setting up `$NMS_HOME/etc/system.`dat, it is important to note that the examples above were presented only for illustration purposes. Parameters may differ on an actual project setting. Coordinate with your Project Engineer in setting up your system configuration file. Also, take note that in the example above, it is assumed that the Generic WebSphere MQ Adapter will reside on the same machine where the Oracle Utilities Network Management System environment resides.

Generic WebSphere MQ Adapter Command Line Options

Command Line Option	Arguments	Required (Y/N)	Description
-areasummary		N	Enables the adapter to process area summary requests. The specified control zone level will be used to filter the outage list in the area summary. The default control zone level of 3 will be used when none is specified.

Command Line Option	Arguments	Required (Y/N)	Description
-complete	Minutes	N	Specifies how often the interface will save a timestamp which is used for synchronization on restart. The timestamp is used to go back to that timestamp to know how far back to get events on restart.
-conditions		N	Enables the adapter to process condition data.
-connect		N	Enables the adapter to process customer disconnect / reconnect information.
-createincident		N	Enables the adapter to process trouble calls.
-crewoutagestatus			Enables the adapter to process crew outage status changes.
-crewupdate		N	Enables crew assignments & dispatches to be sent.
-custdbsname	DBService name	N	Indicates mqseriesgateway to use separate DBService.
-custhistory		N	Enables the adapter to process customer outage history information.
-custstatus		N	Enables the adapter to process customer outage status information.
-customer		N	Enables the adapter to modify the customer model.
-custparseonly		N	Indicates that mqseriesgateway will only parse the customer update xml messages without modifying in the database. This is for performance testing purposes.
-custsqlbundle		N	Enables the adapter to bundle all sql statements for DELETECREATE/CUSTOMER requests to improve speed.
-custstatus		N	Enables the adapter to process customer outage status information.
-debug		N	Enables the adapter to write to standard output the debug information.
-defaultaccounttype	Account Type	N	Sets the default account type for customer model.
-getqueue	Queue name	N	Changes the default get queue name.
-help		N	Writes to standard output, the usage of the adapter.

Command Line Option	Arguments	Required (Y/N)	Description
-includeincident		N	Places incidents information in <PostSrsOutput_001> message. This option is not recommended, because it will greatly degrade the performance of the adapter.
-includepicklistinfo		N	Includes picklist info in the SRS output message if it exists.
-nocompleteevent		N	This option should be used with the –synchronize option. It will exclude all complete event from the synchronization messages
-nosndlist		N	Removes the supply node list from the <PostSrsOutput_001> and <PostSrsOutputStatus_001> messages to improve performance.
-nosndrecache		N	This command line will disable the JMService supply node recache.
-outageupdate		N	Enables outage updates to be sent.
-putqueue	Queue name	N	Changes the default put queue name.
-query		N	Enables the adapter to process query statements to the Operations Database.
-queuemanager	<string queue manager name>	N	Enables the adapter to use the defined queue manager name instead of NMS_MGR.
-recache	<int hours>	N	Enables the adapter to recache SRSService for the customer model.
-recachehour	<hours>	N	This specifies when JMService will re-cache customer info in hours. This will override the –recache option.
-recacheminute	<minutes>	N	This specifies when JMService will re-cache customer info in minutes. This will override the –recache option.
-requestedcblist		N	Enables the adapter to send restore messages containing a list of customers who requested a callback of a restored device.
-sql		N	Enables the adapter to process sql statements to the Operations Database.
-srsoutput		N	Enables the adapter to process outage status (Restored, complete, cancelled).

Command Line Option	Arguments	Required (Y/N)	Description
-subsetcblist	<percent affected> <minimum affected> <maximum affected>	N	This specifies that the Generic WebSphere MQ Adapter will send an SRSOutput message that contains a callback list of relatively random sampling of customers downstream of the restored device. Default values: percent affected=30%, minimum affected=10, maximum affected=300. Setting the percent affected alone is valid but setting the maximum affected needs all three parameters to be present.
-synchronize		N	Sends synchronization messages when the adapter starts. This is only required to capture outage event update messages for events that were completed while the interface was down.
-usecasenotes		N	Supplies case note information with outage messages.
-usedeviceid		N	Enables the adapter to process device names instead of supply nodes index or premise identifiers.
-useincidents		N	Includes a list of incidents with the outage messages.
-usepremiseid		N	Enables the adapter to process premise identifiers instead of supply nodes.
-xslpath	<string path>	N	Indicates that Style Sheet Processing is used, and the literal path the directory that contains the .xsl files for the adapter.

The following table lists and describes command line options to support the bulk load for the customer model createSql process.

Command Line Option	Arguments	Required (Y/N)	Description
-xmlfile	<string path>	N	Indicates the name of the XML file to be used to create the SQL file to produce the customer model.
-writetodb	<>	N	Enables the adapter to write the SQL statements directly to the database when generating the SQL file. (Note when this option is used only a small portion of customers should be used because the entire SQL statement is run.
-outputfile	<string path>	N	Indicates the name of the SQL file to be produced, containing the SQL to create the customer model.

Optionally Configure the Adapter to Run with another Instance of DBService

In `$NMS_HOME/etc/system.dat`, include the MQDBService as one of the services. Use the TCDBService entries as examples of how to set this up.

> **Note**: If using a separate DBService, you must start the Generic WebSphere MQ Adapter with the "-custdbsname" command line parameter and use the MQDBService name as the argument.

Configure the WebSphere MQ Server

- References to "Console Root" throughout this chapter refer to the highest level in the tree displayed by the WebSphere MQ Explorer GUI.

Create New Queue Manager

1. From WebSphere MQ Explorer tree, select:

   Console Root ==> WebSphere MQ ==> Queue Managers ==> New ==> Queue Manager

   - Queue Manager (name) = NMS_MGR.A
   - Check "Make this the default Queue Manager" (indicating yes)
2. Click **Next** – use default settings (circular logging)
3. Click **Next** – use default settings (start queue manager)
4. Click **Next** – uncheck "create listener configured for TCP/IP"

Create New Queues (2)

1. From WebSphere MQ Explorer tree, select:

```
Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A
==>Queues ==> New ==> Local Queue
```

   - Queue Name = NMS.A.FROMNMS
2. Click **OK** – use all default settings
3. Click Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Queues ==> New ==> Local Queue
   - Queue Name = NMS.A.TONMS
4. Click **OK** – use all default settings

   > **Note:** At this point, the two new queues should be created. Check the status of each queue or put a test message into each queue by doing the following:
5. Select Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Queues

   This should display a list of queues.
6. Right click on the desired queue to bring up a menu containing selections for "Status" and "Put test message".

Create Server Connection Channel

1. From WebSphere MQ Explorer tree, select:

```
Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==>
Advanced ==> Channels ==> New ==> Server Connection Channel
```

   A dialog will display containing tabs for General, Extended, MCA, Exits, and SSL

   - In the General tab, the Channel Name is SCH1

- In the MCA tab, the MCA User ID is the local login userid

2. Click **OK** – use all default settings

   Note: At this point, the new server connection channel should be created.
   Check the status of the new server connection channel by doing the following:

3. Select Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Advanced ==> Channels.

   This should display a list of connection channels.

4. Right-click on the SCH1 channel to bring up a menu containing selections for “Status” and “Start” and “Stop”.

5. Select **Start**. The new server connection channel should display the message “The request to start the channel was accepted (amq4008)”.

Create New Queue Manager Listener Service

The default TCP/IP port for the default Queue Manager listener is 1414. Multiple listeners can be configured, but for simplicity, in this case, the original installation default listener for the default queue manager has been stopped. This frees up port 1414 for use by a new listener.

Stopping the original default Q manager listener

1. Select Console Root ==> WebSphere MQ Services (local) ==> (*the original default queue manager name*).

   This will cause a list of services to be displayed, one of which is the “listener” service.

2. Right-click on “listener ==> properties” and stop the listener.

3. Change the startup from Automatic to Manual.

   This listener should no longer start-up at reboot.

Create new Q manager listener for new Q manager

1. Select Console Root ==> WebSphere MQ Services (local) ==> NMS_MGR.A ==> New ==> Listener.

   This will invoke a dialog to create a new “listener” service. This dialog will have three tabs, General, Recovery, and Parameters.

   - The Parameters tab port number must be 1414.
   - The General tab startup type should be “Automatic”.

2. Click the **Start** button on the General tab.

3. To check the status of the listener, select Console Root ==> WebSphere MQ Services (local) ==> NMS_MGR.A ==> Listener ==> Properties

Configure the MQ Client

Set environment variables

The environment configuration file (nms.rc), which is a data file listing Oracle Utilities Network Management System environment settings, should have the following:

```
export MQSERVER=SCH1/TCP/10.115.3.85
```

The environment configuration file must also have two variables set to locate the .TAB file for WebSphere MQ. The .TAB must be copied to the MQ client from the MQ server host as specified by these variables.

Examples:

```
export MQCHLLIB=/users/proj/MQ
export MQCHLTAB=AMQCLCHL.TAB
```

To review IBM's documentation on the MQSERVER environment variable, click on the following URL:

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzal.doc/csqzal10203.htm

View this environment variable (to ensure that it's correct) by typing in the following command: echo $MQSERVER

Test the connection between MQ Client and MQ Server

Test the server connection channel (amqscnxc)

On the Unix command line, type in the following command:

```
/usr/mqm/samp/bin/amqscnxc -x 10.115.3.85 -c SCH1 NMS_MGR.A
```

where:

- -x is the IP address of the MQ Server host
- -c is the Server Connection Channel Name
- the third parameter is the desired Queue Manager Name

Test 'putting' a message from Server to Client (amqsputc)

On the Unix command line, type in the following command:

```
/usr/mqm/samp/bin/amqsputc NMS.A.FROMNMS Sample AMQSPUT0 start
target queue is NMS.A.FROMNMS
```

```
<MSG-FROM-SVR>VOILA</MSG-FROM-SVR> Sample AMQSPUT0 end
```

The message should appear in the queue named NMS.A.FROMNMS which can be viewed on the client using the MQ Explorer GUI at:

```
Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==>
Queues ==> NMS.A.FROMNMS ==> Browse Messages
```

Test 'getting' a message on Client from Server (amqsgetc)

First "get" the message just written

```
/usr/mqm/samp/bin/amqsgetc NMS.A.FROMNMS Sample AMQSGET0 start message
<<MSG-FROM-SVR>VOILA</MSG-FROM-SVR>> no more messages Sample AMQSGET0
end
```

Test 'putting' a message from Client to Server using WebSphere MQ GUI

1. Select Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Queues ==> NMS.A.TONMS ==> Put Test Message
2. Paste the following into "Message Data":

   ```
   <MSG-FROM-CLNT>VOILA</MSG-FROM-CLNT>
   ```

3. Click **OK**.

   The following message should be displayed: "The test message was put successfully (amq4016)".

Test ‘getting’ a message on Server from Client (amqsgetc)

On the Unix command line:

```
/usr/mqm/samp/bin/amqsgetc NMS.A.TONMS Sample AMQSGET0 start message
<<MSG-FROM-CLNT>VOILA</MSG-FROM-CLNT>> no more messages Sample
AMQSGET0 end
```

High Availability

The goal of the Oracle Utilities Network Management System MQ XML adapter redundancy is to provide assured message receipt and delivery between the Oracle Utilities Network Management System services and the WebSphere MQ queues. There are a number of availability approaches that could be utilized, with potentially different approaches being used for each system to be interfaced. The purpose of this section is to describe different availability approaches that can be used with the adapter.

Clustering

One approach for high availability is to utilize WebSphere MQ clustering on Windows where persistent queues would be used. The adapter is then responsible for maintaining connection to InterSys. All messages would be persisted to one physical disk location, where either redundant Oracle Utilities Network Management System Generic WebSphere MQ Adapter would have access to the same message.

Non-Redundant Queue Approach

This would be a classical implementation using WebSphere MQ. A single copy of the Oracle Utilities Network Management System Generic WebSphere MQ Adapter would be used to retrieve information from a single (non-redundant) set of message queues.

This approach is the simplest and most common way to implement WebSphere MQ queues for an application. However, a hard failure of the adapter or its server could result in message losses.

Synchronization Process

In the case in which both adapter servers are down, the Generic WebSphere MQ Adapter provides a synchronization process on startup by specifying `-synchronize` command line option. This will retrieve all events information from OPS and put them on the queue with topic_type = ‘TRBL_UPDATE’ in <PostSrsOutput_001> message with <srsOutputMsgType>1</srsOutputMsgType> and <description>SYNCHRONIZE</description>. This synchronization can also be triggered again by issuing the following command from the UNIX server command prompt:

```
Action any.mq* synchronize
```

The synchronized messages will include the latest status of all active events and completed/ cancelled events in the past N days, where N is the number of days since the adapter was last running.

Troubleshooting

High-level messages are typically used within Oracle Utilities Network Management System to permit one process to control another process. There are no special high-level messages that would be required for this adapter.

Note that doing an Action any.any stop will stop the adapter, which needs to be taken into consideration for administering the adapter when starting and stopping it.

Supported high level messages include the following:

- report
- debug <debug level>
- stop

Example usage: `Action any.mqseriesgateway report`

Performance

This interface is intended to provide for high performance as needed to process frequent message exchange such as in the case of trouble calls during a storm. In order to provide optimum performance, there are aspects of both implementation and usage. Aspects of usage include:

1. Sending multiple trouble calls together in a single message when possible will improve the call volume.
2. Providing a valid device identifier (supply node) on each trouble call.
3. Avoiding unnecessary queries within Oracle Utilities Network Management System, which might otherwise degrade overall system performance either due to locking, CPU utilization and/or disk access.
4. Using a customer account (service account), or premiseId as apposed to the custDeviceIdx with a supply node, will slow down the overall performance of trouble call processing and potentially outage prediction.

The following aspects of implementation can optimize performance:

It is assumed that incoming XML messages are well formed, bypassing the validation step. It is assumed that the sender provided well-formed XML, which was transmitted using reliable communication mechanisms. The actual validation test is whether or not the code that internally parses a message can extract a sufficient set of parameters to make an InterSys request. XML that is not well formed will typically generate an error. It should also be noted that XML validation does not necessarily guarantee valid information provided by an external system. If this generates an error in the adapter, it will be generate an error.

Use of multiple threads within the adapter, permitting parallel processing. This makes it possible to process multiple requests concurrently, providing the potential to increase performance.

Functional Requirements

The purpose of this section is to describe the desired functionality. Key requirements include the following:

- Get customer(s) query using fuzzy keys (as through call entry application).
- Get customer outage status information, which would indicate whether or not a customer is part of an existing outage.
- Get customer outage history, which would provide the available outage history for a customer who may have been part of an old outage event.
- Create incident, to send trouble calls to Oracle Utilities Network Management System for outage analysis.
- Add, remove, and update condition. One example of possible usage would be to get tags from SCADA for display within Oracle Utilities Network Management System.
- Get conditions for a specified device as maintained within Oracle Utilities Network Management System.

- Create, Delete, and Update customer information, as might typically be used by a Customer Information System (CIS) to maintain customer information within Oracle Utilities Network Management System. Support a mechanism that will provide a means of generating a large number of customer transactions to initially load the customer model.
- Check interface/InterSys status, permitting an external system to see whether or not the adapter is currently active and whether or not Oracle Utilities Network Management System is active.
- Report Current outage status to indicate current state of an outage.
- Report Crew outage states to indicate crew events that may change the state of an outage in Oracle Utilities Network Management System.
- Receive customer disconnect and reconnect indications to identify if the utility has purposely disconnect power or restored power for a customer.
- Network trace including planned outage and current feeder request.
- SQL Queries and Database transactions to get reports or manipulate specific customer specific tables.
- Support for style sheet translation on incoming and outgoing XML messages.
- Support for external notification when the status of a device changes such as open or close.
- Synchronize outage status between Oracle Utilities Network Management System and external system.
- Create area summary list
- Create callback list

Design Overview

The general approach is to build an adapter process that passes messaging between the ISIS and WebSphere MQ messaging systems. Messages sent and received using WebSphere MQ will be formatted in XML. The types of messaging that will be supported include the following:

- Asynchronous publish from Oracle Utilities Network Management System to WebSphere MQ (using ‘fire and forget’ pattern)
- Asynchronous publish from WebSphere MQ to Oracle Utilities Network Management System (using ‘fire and forget’ pattern)
- Request/reply from WebSphere MQ to Oracle Utilities Network Management System (the requestor can process this either synchronously or asynchronously)

As the Oracle Utilities Network Management System currently supports a large number of messages internally, the principle of this interface is to externalize support for a key subset of those messages through the WebSphere MQ adapter.

Data Flows

Overview

The purpose of this section is to describe the information flows that are relevant to this interface. The details of each of these flows and associated XML formats are provided in the appendices.

All flows are formatted using XML, with specific XML definitions defined as to be consistent with Oracle Utilities Network Management System product direction. XML messages will provide well-formed XML, without a requirement for validation, as validity will be the responsibility of the sender. Consequentially, there is also no requirement for the specification of DTDs or XML Schema definitions. Specific XML requirements are defined in the appendixes of this document.

Create Incident

This is an asynchronous input to the adapter to report trouble calls. Internally the SRS::sendSRSinput method is used to send the trouble call to Oracle Utilities Network Management System. This interface supports a variety data that might be input on a trouble call.

This will be sent using an asynchronous fire and forget.

Adapter can also process fuzzy call with street intersection data. This requires att_street_segment and att_street_intersect tables to be populated.

Note: Performance is optimized when the customer supply node is identified.

Get Customer Outage Status

This is a request/reply interface that is used to obtain the current status of a customer. The information returned includes:

- Whether or not the customer is part of an existing outage
- Estimated restoration time
- Whether or not customer is part of a planned outage

This will be processed using an asynchronous inquiry. It should be noted that several queries are required to collect this information, it will degrade overall performance in storm situation.

Get Customer Outage History

This is a request/reply interface that is used to obtain customer call and outage history.

- Call history
- Outage history

This message will be processed using an asynchronous inquiry. It should be noted that several queries are required to collect this information, it will degrade the overall performance in storm situation. This history is supply based on the current Ops database, and will not get history from any historical database.

Create, Delete, Update, Get Condition

This interface would be used to report tags and other types of conditions. Conditions are objects that describe and manage important information associated with objects that are defined within the Oracle Utilities Network Management System model. Subclasses of the “condition” class include “tag” and “note”, or any other configured condition in Oracle Utilities Network Management System.

This message is an asynchronous input to the adapter.

Outage Status

This interface is an asynchronous output from the adapter used to publish SRS output messages. SRS output messages describe creation, update, and closure of information related to outages. If TRBL_ERT_UPDATE message type is configured in the mq_adapter_config table, then global ERT changes from Stormman will trigger <postSrsOutput_001> message with type = 20 to be published for each event. For performance considerations, type = 20 message only has limited information.

Create, Delete, Update, Get Customer

This is an asynchronous input interface used to report updates to the customer data model. These updates may be the addition, modification or deletion of a customer. To get information for an existing customer, the get verb is supported in order to retrieve this information. In the event of an update, only the modified information needs to be supplied. Internally this will use the DBS::sql method to perform updates to the appropriate tables that define the customer data model. It is assumed that CES_CUSTOMER and CUSTOMER_SUM are implemented as views, and the Oracle Utilities Network Management System MultiSpeak-based customer model is utilized.

This interface is implemented in a manner to provide extensions over the capabilities of MultiSpeak, including multiple meters and/or transformers for a service location. The interface uses the internal service point entity to manage these relationships.

Customer update can be configured to use separate DBService by using command line option –`custdbsname` <*DBSNAME*> to improve overall performance. This requires the server side to have an instance of DBService reserved for this purpose.

There is a new command line option available –custsqlbundle which will greatly improve the performance for DELETECREATE/CUSTOMER request.

> **Note:** It will be necessary to have JMService update its internal cache periodically. This will be triggered by the adapter nightly by the defined recache period that is supplied to the adapter. The default value is 24 hours and can be changed through `–recache` <*HOURS*>. Re-cache can also be configured to run at specific hour and minute by specifying `–recachehour` <*HOUR*> and `-recacheminute` <*MINUTE*>. These two command line options will override `–recache` <*HOURS*> option. The recache function can be disabled by using `–nosndrecache` option.

SQL Transactions

This is an asynchronous input interface is used to send SQL transactions to Oracle Utilities Network Management System. This internally uses the DBS::sql method. This interface is activated by the – sql command line option.

> **WARNING:** This interface, if activated, is a potential security issue in non-trusted environments, as it would be possible to execute destructive transactions against the Oracle Utilities Network Management System database.

> **WARNING:** This interface, if activated and used inappropriately, can be a source of system performance degradation or denial of service. This would be the case if long duration transactions were run against the Oracle Utilities Network Management System database, especially if done against key Oracle Utilities Network Management System tables.

SQL Query

This is a request/reply interface that is used to perform an SQL query on Oracle Utilities Network Management System and return the resulting selection set. The DBS::query method is used internally. The returned XML is formatted using appropriate tags, using column names, and row delimiters.

> **WARNING:** This interface, if activated and used inappropriately, can be a source of system performance degradation or denial of service. This would be the case if long duration queries were run against the Oracle Utilities Network Management System database, especially if done against key Oracle Utilities Network Management System tables.

Status Check

This is a request/reply interface that is used to check the status of the Generic WebSphere MQ Adapter.

Errors

Errors detected will be asynchronously reported on the defined error queue. An error queue will be as define per the reply queue for any out going message. If there is no reply queue, the default queue will be used to supply the error messages to.

Customer Disconnect / Reconnect.

This information flow is implemented as an asynchronous request for disconnecting or reconnecting customers, indicating which customers have had power disconnected or reconnected by the utility. The customers who have been disconnected will not be seen by Oracle Utilities Network Management System call taking applications, as these customers have purposely been disconnected by the utility for payment reasons. In order for these customers to be ignored from call taking applications that will be using the CreateIncident flow defined in this document, the checkCutoff flag for customers that call in should be used for JMService to be notified to check for the disconnected customers. If these systems know that the customers are disconnected, calls for these customers only need to use the checkCutoff flag.

Crew Outage Status Changes

The purpose of this section is to provide crew states that may or may not transition an outage state change in Oracle Utilities Network Management System. Oracle Utilities Network Management System currently provides crew state changes for an outage via a CrewMessage. The CrewMessage contains the following message types for crew messages. Each message type identified below also provides whether or no the message could change the status of the outage.

Crew Message Type	Classes APIs available in	Potentially generates outage state change	Description.
INVALID	None	NO	Invalid message was sent. Unsupported message for all crew APIs
SET_CONST	None	NO	Sets static constant(s) for the Crew class, which will make non-standard functionality of the crew classes be supported.
CLEAR_CONST	None	NO	Remove a static constant that has been set. This may make some supported functionality of the crew classes' disabled.
AVAIL_FOR_OP	Crew	NO	
UNAVAIL_FOR_OP	Crew	NO	
CREATED	Crew	NO	
DELETED	Crew	NO	
EDITED	Crew	NO	
ACTIVATED	Crew	NO	

Crew Message Type	Classes APIs available in	Potentially generates outage state change	Description.
DEACTIVATED	Crew	NO	
ASSIGNED	CrewAssignment	YES	
UNASSIGNED	CrewAssignment	YES	
DISPATCHED	CrewDispatch	YES	
UNDISPATCHED	CrewDispatch	YES	
RELOCATE	CrewDispatch	YES	
AVAILABLE	Crew	NO	
UNAVAILABLE	Crew	NO	
ASSIGNMENT_CHANGED	CrewAssignment	YES	
DISPATCH_CHANGED	CrewAssignment	YES	
ARRIVED	CrewDispatch	YES	
UNARRIVED	CrewDispatch	YES	
TEMP_ZONE_CHANGE	Crew	NO	
UPDATE_SCHEDULE	Crew	NO	
SUSPENDED	CrewAssignment	YES	
CASE_NOTES_INFO_CHANGED	Crew	NO	
CREW_REQUEST_ADD	CrewRequest	NO	
CREW_REQUEST_EDIT	CrewRequest	NO	
CREW_REQUEST_DELETE	CrewRequest	NO	

These changes may affect the status of the outage, or just the status of a crew. Via configuration of the mq_adapter_config table, any crew state change may trigger the creation of a PostSrsOutputStatus_001 XML message. This message will be identical in format to the PostSrsOutput_001 XML message, except that it will be triggered by the configure crew message type as previously defined. If the configuration for the crew message type is made, any queue can be used to put the PostSrsOutputStatus_001 message in.

Note: The crew message may not affect the current state transition of the outage, and may provide redundant data. In order to avoid un-necessary overhead of providing SRSOutput status information for non-required crew states, it is highly recommended that only the minimal crew states that are needed for the outage state changes are used in order to get the PostSrsOutputStatus_001 XML message. This flow will have to go to JMService to regenerate the outage information.

An example for configuring the receipt of a PostSrsOutputStatus_001 is provided below.

```
INSERT INTO mq_adapter_config VALUES ('OMS_OUTAGE_STATUS', 2,
                                      'PostSrsOutputStatus_001',
                                      'DISPATCHED',
                                      '', '', 1);
```

In this example, if a message type of "DISPATCHED" is received by the adapter, SRS will (via SRS::getEventInfo()) be requested for the current event status of the outage. The outage status can be retrieved via the Crew with the current outage data; a PostSrsOutputStatus_001 XML message will be generated to send to the OMS_OUTAGE_STATUS queue. Note that any queue can be defined to publish the Outage Status to. This will be configurable based on the number of configuration parameters for the PostSrsOutputStatus_001 XML message. Whichever crew message types should trigger the creation of the XML message, each one (message type and put queue) will need to be identified and defined in the mq_adapter_config table. Please refer to PostSrsOutput_001 XML message for the data content of the PostSrsOutputStatus_001 message.

The purpose of this information flow is to provide crew outage status messages when ever a crew is Dispatched, Un-Dispatched, Assigned, or Un-Assigned from an event. In order to provide the crew outage status change for the events, this flow is required. This will be an asynchronous event that will provide the crew outage status changes to any interface that may require this information.

Sending Crew Updates / Getting Crew (Request / Reply) Information

When an external system needs to get crew details for one Crew, or all Crews, or the Crew needs to be indicated assigned, un-assigned, dispatched, un-dispatched, arrived, available, unavailable, active, in-active, or suspended from an event in or independent from an event in Oracle Utilities Network Management System; the external system will send a crew update message with the event id to Oracle Utilities Network Management System. The VERB of the CrewUpdate_001 XML message will indicate the state or action to be triggered by the message. Creating, deleting and editing crews is also supported by this flow.

Published Crew Information Updates to an External System

When a crew is assigned, un-assigned, dispatched, un-dispatched, arrived, suspended, created, deleted, or edited in Oracle Utilities Network Management System; the system will broadcast this information to any external system that would be interested in receiving this information about the crew.

Network Trace Includes Planned Outage Request and Current Feeder Request

Oracle Utilities Network Management System has the ability to send all supply nodes back for a specific device (planned outage request). Oracle Utilities Network Management System can also send back feeder, substation and control zone information back for a specific supply node (current feeder request). To enable this function, one line needs to be added to MQ_ADAPTER_CONFIG table and specify '-connectivitytrace' command line option:

```
insert into mq_adapter_config values ('REQUEST_TRACE',3,
'TraceNetwork_001','TraceNetwork_001',null,'REPLY_TRACE',1);
```

Area Summary

The MQ Adapter will respond to a request for an area summary. The reply message will contain information that is available in the Performance Mart database. A synonym will be created to make this data available in the Ops database. The MQ Series Adapter will query this table and send back the information.

This will be a request/reply data flow. The external system must request an outage summary for the MQ Adapter to send a response. The areas that are to be summarized will exactly match some level(s) as defined in the Performance Mart database. The areas that are returned will be the Area Control Zones with recognized active outages. The control zone level of the included areas can be configured through the command line.

Callback List

The callback list message will use the same body as the Post SRSoutput message. It will have a different noun and verb.

There are two types of callback list. One contains in the incident list all of the callers who have requested a callback. This message will have a verb of REQUESTED and a noun of CALLBACKLIST. The second type of callback list is a relatively random sampling of customers downstream of the restored device. This message will have a verb of SUBSET and a noun of CALLBACKLIST.

There are configurable parameters that will set the maximum and minimum percentage of affected customers who will be included in the SUBSET callback list. The maximumCallbackSample and minimumCallbackSample for the subset callback can be set through the srs_rules table. These parameters will default to 50 and 4 percent respectively if not configured explicitly. Both types of callback lists use the same format found in PostSrsOutput_001.xsd.

Information Model

This section provides an overview of the logical information model supported by this interface. The key objects supported by this interface include:

- Customers, which are defined using accounts, service locations and meters. This model is based upon the MultiSpeak model. Typically, as a practical note, the custId identifier may in fact be the same as the account number. Some extensions to the MultiSpeak model are used as required to address issues that are otherwise not addressed by MultiSpeak. The support of a bulk load process that reads an XML file, with defined customers to create the model. This process (createSql) can be run to generate the SQL to be run on the production servers, or can directly create the customers.
- Trouble calls are also referred to as incidents with in Oracle Utilities Network Management System. An incident is typically related to a customer, who in turn is related to a device. In the absence of a correlation to a device, a trouble call is classified as a 'fuzzy' call, which differentiates it from a call that can be directly correlated to the electrical distribution network.
- Outages, which are a consequence of the correlation of incidents. Outages are one form of an event that is managed by JMService. Some events are non-outage events, such as power quality. The type of call that is provided can identify such non-outage and outage events. Each call needs to be identified with a trouble code, which will determine the type of call that JMService will generate with in Oracle Utilities Network Management System.
- Devices, which are part of the electrical distribution network. Customers, outages and conditions may have relationships to devices. Typically customers are related to transformer devices. Outages are typically related to switch, fuse or transformer devices.
- Conditions (which can be specialized within Oracle Utilities Network Management System for the management of information such as tags, notes, etc.)

- SQL queries, result sets and transactions.
- Customer disconnections and reconnections for indicating customers who have been purposely removed from Service by the utility.
- Crew Outage States that will identify outage states that change as a result of a crew action. For example a crew that has been dispatched, assigned, or suspended from outage work would correlate to an action that may trigger an outage state change in Oracle Utilities Network Management System.

The information described by these models is formatted using XML for the purposes of exchange through this interface. The following table describes tags that are used in the XML definitions, and how they relate to the information model within Oracle Utilities Network Management System. The corresponding types used in these models are I = Integer, S = String, T = TimeStamp, C = Single Character, and F = floating point.

SRS Output and SRS Output Status Message Tags

Tag	Parent entity	Description	Type
PostSrsOutput	PostSrsOutput_001	The start of the SRS Output message	S
PostSrsOutputStatus	PostSrsOutputStatus_001	The start of the crew outage status message.	S

Tag	Parent entity	Description	Type
srsOutputMsgType	PostSrsOutput, PostSrsOutputStatus, outage	Type of SRS output message: 1=Create outage condition and alarm 2=Remove outage condition and alarm 3=Change outage condition or alarm 4=Send a fuzzy alarm 5=Create incident 6=Clear incident 7=Priority Call groups to existing DO 8=A Pending Cancel 9=Reschedule Meet or Outage 10=TEService internal messages 11=Send an unassigned alarm 12=Remove all "incident" alarms for an event 13=crew has been removed AND the update trouble button on the picklist has been selected 14=update trouble button on the picklist has been selected 15=this message tells the viewer not to display the outage any more 16=message intended to invoke case note deletion 17=the secondary SRS is now up and running 18=clear case note message 19=event action added or deleted 20=Estimated assessment/restore time values updated 21=damage report updated or created 22=update callback information 23=no type 99=Trouble Clear	I
district	PostSrsOutput, PostSrsOutputStatus, outage	District name of the outage device	S
office	PostSrsOutput, PostSrsOutputStatus, outage	Office name of the outage device	S
circuit	PostSrsOutput, PostSrsOutputStatus, outage	Circuit name of the outage device	S
feeder	PostSrsOutput, PostSrsOutputStatus, outage	Feeder name of the outage device	S
Ncg	PostSrsOutput, PostSrsOutputStatus, outage	ID of the control zone of the outage device	I
partition	PostSrsOutput, PostSrsOutputStatus, outage	Partition ID typically identifies the map sheet on which a device is associated	I
appliedRule	PostSrsOutput, PostSrsOutputStatus, outage	Rule number used by SRS to determine outage device	I
numb	PostSrsOutput, PostSrsOutputStatus, outage	Internal identifier for each outage record	I

Tag	Parent entity	Description	Type
ruleSet	PostSrsOutput, PostSrsOutputStatus, outage	Name of active rule set used by SRS to process outage	S
eventCls	PostSrsOutput, PostSrsOutputStatus, outage	Event class. Class 800 indicates an outage event. Part of the event handle.	I
eventIdx	PostSrsOutput, PostSrsOutputStatus, outage	Event index, part of the event handle that uniquely identifies event in conjunction with the eventCls.	I
eventApp	PostSrsOutput, PostSrsOutputStatus, outage	Event app, part of the event handle that uniquely identifies event in conjunction with the eventCls and eventIdx.	I
alarmCls	PostSrsOutput, PostSrsOutputStatus, outage	Alarm class, part of alarm handle. Used to uniquely identify rows on work agenda	I
alarmIdx	PostSrsOutput, PostSrsOutputStatus, outage	Alarm index, part of alarm handle. Used to uniquely identify rows on work agenda	I
deviceCls	PostSrsOutput, PostSrsOutputStatus, outage	Class or outage device. Identifies the type of device that failed.	I
deviceIdx	PostSrsOutput, PostSrsOutputStatus, outage	Used in conjunction with the deviceCls to form the device handle, which uniquely identifies a device.	I
deviceAlias	PostSrsOutput, PostSrsOutputStatus, outage	Name of outage device, as defined in the ALIAS_MAPPING table.	S
deviceApp	PostSrsOutput, PostSrsOutputStatus, outage	Used in conjunction with the deviceCls and deviceIdx to form the device handle, which uniquely identifies a device.	device App
cause	PostSrsOutput, PostSrsOutputStatus, outage	Outage cause (TROUBLE_CALL, FAULT_INDICATOR or blank)	S
description	PostSrsOutput, PostSrsOutputStatus, outage	Short description based on status	S
troubleCode	PostSrsOutput, PostSrsOutputStatus, outage	Entered trouble code or combination from all calls	S
troubleQueue	PostSrsOutput, PostSrsOutputStatus, outage	Dispatcher queue for outage	S
status	PostSrsOutput, PostSrsOutputStatus, outage	Alarm state name	S
operatorComment	PostSrsOutput, PostSrsOutputStatus, outage	Operator entered comment	S
tags	PostSrsOutput, PostSrsOutputStatus, outage	Y if tags exist N if no tags exist X if tags are not checked	C

Tag	Parent entity	Description	Type
estSource	PostSrsOutput, PostSrsOutputStatus, outage	Source of estimated restoration time: C=crew (manually entered) S=Storm Management (regular) G=Storm Management globally applied ERT P=Storm Management non-published ERT O=Storm Management (crew dispatched/ onsite) N=none	C
externalId	PostSrsOutput, PostSrsOutputStatus	Identifier for event supplied by an external system.	I
crewId	PostSrsOutput, PostSrsOutputStatus, outage	Crew(s) assigned to outage	I
firstIncTime	PostSrsOutput, PostSrsOutputStatus, outage	Time of first reported incident.	T
firstCrewTime	PostSrsOutput, PostSrsOutputStatus, outage	Time of first dispatched crew.	T
crewOnSiteTime	PostSrsOutputStatus	Time of crew arrival	T
estRestTime	PostSrsOutput, PostSrsOutputStatus, outage	Estimated restoration time.	T
outageTime	PostSrsOutput, PostSrsOutputStatus, outage	Time the outage began.	T
jobCompletionTime	PostSrsOutput, PostSrsOutputStatus, outage	Time outage was completed.	T
restoreTime	PostSrsOutput, PostSrsOutputStatus, outage	Time the outage was restored.	T

Tag	Parent entity	Description	Type
srsCondStatus	PostSrsOutput, PostSrsOutputStatus, outage	Outage status codes: 0=no outage 1=probable service outage 2=probable device outage 3=real service outage 4=real device outage 7=non outage 8=critical meet 9=future meet 10=confirmed service outage 11=confirmed secondary outage 12=additional alarm 13=probable momentary outage 14=real momentary outage 15=planned outage	I
condPhases	PostSrsOutput, PostSrsOutputStatus, outage	Bitmask to identify affected phases	I
customersOut	PostSrsOutput, PostSrsOutputStatus, outage	Number of customers affected by an outage.	I
srsPriority	PostSrsOutput, PostSrsOutputStatus, outage	Number of priority calls, may be redefined by configuration	I
custCall	PostSrsOutput, PostSrsOutputStatus, outage	Number of customers affected by an outage that have called.	I
priW	PostSrsOutput, PostSrsOutputStatus, outage	Number of wire related calls, may be redefined by configuration	I
priSW	PostSrsOutput, PostSrsOutputStatus, outage	Number of service wire related calls, may be redefined by configuration	I
priP	PostSrsOutput, PostSrsOutputStatus, outage	Number of pole problem calls, may be redefined by configuration	I
priE	PostSrsOutput, PostSrsOutputStatus, outage	Number of emergency calls, may be redefined by configuration	I
custCrit	PostSrsOutput, PostSrsOutputStatus, outage	Number of affected critical customers	I
crit1	PostSrsOutput, PostSrsOutputStatus, outage	Number of category 1 customers that called, as determined by configuration	I
crit2	PostSrsOutput, PostSrsOutputStatus, outage	Number of category 2 customers that called, as determined by configuration	I

Tag	Parent entity	Description	Type
crit3	PostSrsOutput, PostSrsOutputStatus, outage	Number of category 3 customers that called, determined by configuration	I
critK	PostSrsOutput, PostSrsOutputStatus, outage	Affected category K (KEY) customer, may be redefined by configuration	I
critC	PostSrsOutput, PostSrsOutputStatus, outage	Affected category C (CRITICAL) customer, may be redefined by configuration	I
critD	PostSrsOutput, PostSrsOutputStatus, outage	Affected category D customer, may be redefined by configuration	I
critTot	PostSrsOutput, PostSrsOutputStatus, outage	Total number of critical customers, recognizing that a customer is counted only once even though it may belong to more than one category	I
revenue	PostSrsOutput, PostSrsOutputStatus, outage	Revenue of the total customers from customer_sum	revenue
customerName	PostSrsOutput, PostSrsOutputStatus, outage	Used for fuzzy calls	S
addrBuilding	PostSrsOutput, PostSrsOutputStatus, outage	Used for fuzzy calls	S
addrStreet	PostSrsOutput, PostSrsOutputStatus, outage	Used for fuzzy calls	S
addrCity	PostSrsOutput, PostSrsOutputStatus, outage	Used for fuzzy calls	S
customerPhone	PostSrsOutput, PostSrsOutputStatus, outage	Used for fuzzy calls	S
xRef	PostSrsOutput, PostSrsOutputStatus, utage	X reference coordinate	F
yRef	PostSrsOutput, PostSrsOutputStatus, outage	Y reference coordinate	F
sheetNum	PostSrsOutput, PostSrsOutputStatus, outage	Number of switching sheet associated with a planned outage.	I
dispAddress	PostSrsOutput, PostSrsOutputStatus, outage	Dispatch address, first incident if probable service outage, feeder name if no incidents or probable device outage	S
groupType	PostSrsOutput, PostSrsOutputStatus, outage	GRP is manually grouped, REL if related	S
hasClue	PostSrsOutput, PostSrsOutputStatus, outage	Y if one or more incidents for this outage has a clue	C
ctrlZoneName1	PostSrsOutput, PostSrsOutputStatus, outage	Control zone name, level 1	S
ctrlZoneName2	PostSrsOutput, PostSrsOutputStatus, outage	Control zone name, level 2	S

Tag	Parent entity	Description	Type
ctrlZoneName3	PostSrsOutput, PostSrsOutputStatus, outage	Control zone name, level 3	S
ctrlZoneName4	PostSrsOutput, PostSrsOutputStatus, outage	Control zone name, level 4	S
ctrlZoneName5	PostSrsOutput, PostSrsOutputStatus, outage	Control zone name, level 5	S
ctrlZoneName6	PostSrsOutput, PostSrsOutputStatus, outage	Control zone name, level 6	S
devClsName	PostSrsOutput, PostSrsOutputStatus, utage	Name of class of outage device	S
AffectedSupplyNodeList	PostSrsOutput, PostSrsOutputStatus, outage	The affected list of supply nodes	S
snd	AffectedSupplyNodeList	The affected supply node.	I
AffectedDeviceIdList	PostSrsOutput, PostSrsOutputStatus	The indicator for the device ids for the affected transformers.	S
devId	AffectedDeviceIdList	The affected device id for the transformer.	S
whoCompleted	PostSrsOutput, PostSrsOutputStatus, outage	The name of the user who completed the outage.	S
whoResponsible	PostSrsOutput, PostSrsOutputStatus, outage	The name of the user who is currently responsible for the event.	S
whoAcknowledged	PostSrsOutput, PostSrsOutputStatus, outage	The user who acknowledged the event.	S
AffectedPremiseIdList	PostSrsOutput, PostSrsOutputStatus, outage	This list is used as an option for systems that cannot have a transformer relationship to a customer. This is a last resort and is highly recommended not to be used because of performance impacts when using this option	S
premiseId	PostSrsOutput, PostSrsOutputStatus, outage	The customer's premise id, or serv_loc_id in the customer model	S
AffectedDeviceIdList	PostSrsOutput, PostSrsOutputStatus		S
devId	PostSrsOutput, PostSrsOutputStatus		S
CaseNote	PostSrsOutput, PostSrsOutputStatus, Outage		S
caseNoteText	CaseNote	Text for the case note.	s
Note: The following tags appear in <PostSrsOutput_001> message only if the '-includeincident' command line option is used. Oracle recommends against using this option.			
IncidentList	PostSrsOutput, PostSrsOutputStatus, outage		
Incident	IncidentList		

Tag	Parent entity	Description	Type
incTroubleCode	Incident		S
incDevice	Incident		S
incAccount	Incident		S
incCustDeviceCls	Incident		I
incCustDeviceIdx	Incident		I
incCustName	Incident		S
incAddress	Incident		S
incStreet	Incident		S
incCity	Incident		S
incPhone	Incident		S
incIncidentTime	Incident		S
incComment	Incident		S
incCallbackLate	Incident		S
incExternaId	Incident		S
incCallCancel	Incident		S
incLifeSupport	Incident		S
incCustPriority	Incident		S
incMeterId	Incident		S
incGeneralArea	Incident		S
incCustOrderNum	Incident		S
incDrvInst	Incident		S
incCallbackRequest	Incident		S
incCustCritical	Incident		S
incAlternatePhone	Incident		S
incCserName	Incident		S
incXRef	Incident		F
incYRef	Incident		F
incEventCls	Incident		I
incEventIdx	Incident		I

Note: Many values, names and usages are configurable and vary between implementations. The details and description of configuration options are outside the scope of this document.

Note: For performance reasons the type 20(TRBL_ERT_UPDATE) message contains limited information. The available fields are: `<srsOutputMsgType>`, `<eventIdx>`, <estRestTime> and <estSource>. Other fields are either NULL or 0.

Customer Message Tags

Refer to the Oracle Utilities Network Management System Customer Data Model specification.

Note: The bulk load of customers can be accomplished by running the createSql.exe file that will create .sql file to be run through ISQL.ces, which will populate the customer model. The createSql file should be provided the following command line options to create the sql. (IE. CreateSql –xmlfile create_customers.sql –outputfile product_customer_data.sql.) The –writetodb command can be provided to write the customers directly through to the database with out having to run ISQL.ces < product_customer_data.sql.

Trouble Call Message Tags

Tag	Parent	Description
Incident		
troubleCode	Incident	
device	Incident	
account	Incident	
custDeviceCls	Incident	
custDeviceIdx	Incident	
name	Incident	
address	Incident	
city	Incident	
phone	Incident	
incidentTime	Incident	
comment	Incident	
callbackLate	Incident	
externalId	Incident	
callCancel	Incident	Deprecated and project specific
lifeSupport	Incident	
custNone	Incident	
custPriority	Incident	
custPhoneArea	Incident	
callId	Incident	

Tag	Parent	Description
meterId	Incident	
custLastName	Incident	
generalArea	Incident	
custOrderNum	Incident	
drvInst	Incident	
addrBuilding	Incident	
addrStreet	Incident	
addrCity	Incident	This is also used in fuzzy calls to get a better indication of which intersection point should be used.
meetTime	Incident	
meetType	Incident	
cidAlias	Incident	
powerUp	Incident	
cancelCall	Incident	Deprecated and project specific
callbackRequest	Incident	
callbackTime	Incident	
custIntrX	Incident	
custIntrY	Incident	
updateExistingInc	Incident	
custCritical	Incident	
alternatePhone	Incident	
userName	Incident	
checkCutoff	Incident	
xRef	Incident	
yRef	Incident	
custPhoneUpdate	Incident	
streetNameA	Incident	Used to indicate a street name for a fuzzy call
streetNameB	Incident	Used to indicate a street name for a fuzzy call.
addressNumber	Incident	Used for the street number, to find in an address range for it in the range of an intersection point. The street number is <> the range.
addrState	Incident	This is also used in fuzzy calls to get a better indication of which intersection point should be used.
addrPostCode	Incident	This is also used in fuzzy calls to get a better indication of which intersection point should be used.

Tag	Parent	Description
premiseId	Incident	This is the premiseId (serv_loc_id) for the customer. This XML tag should only be used as a last option if supply node or customer account number cannot be used with a call. This field incurs additional performance issues if used, and is highly recommended not to use it as an option for processing trouble calls.
controlZoneName	Incident	If the control zone name is passed with the call, it will be used to append control zone information with the call to be placed at this control zone within JMService. This would be useful in cases where customers are not yet modeled in Oracle Utilities Network Management System, and calls would want to be placed as close to the lowest control zone as possible.
eventCls	Incident	Related event class. Used for canceling or updating outages.
eventIdx	Incident	Related event index. Used for canceling or updating outages.

Crew Message Tags

Tag	Parent	Description
CrewList	DATAAREA	
Crew	CrewList	
crewId	Crew	
crewContact	Crew	
crewType	Crew	
crewPagerNumber	Crew	
crewMobileNumber	Crew	
crewMdt	Crew	
crewSupervisor	Crew	
crewCenter	Crew	
controlZone	Crew	
crewStatus	Crew	
crewVehicleList	Crew	
crewVehicle	crewVehicleList	
crewVehicleId	crewVehicle	
crewVehicleType	crewVehicle	
crewActive	Crew	

Tag	Parent	Description
crewAvailable	Crew	
crewNcgCls	Crew	
crewNcgIdx	Crew	
userName	Crew	
eventCls	Crew	
eventIdx	Crew	
crewMemberList	Crew	
CrewMember	crewMemberList	
crewMemberName	CrewMember	
crewMemberTechId	CrewMember	
crewMemberType	CrewMember	

Crew Outage Status Changes

Refer to the SRS Output Message tags, as the DATA Content tags will be the same for this flow.

Configure Queues for Required Data Flows

Any data flows to be used require WebSphere queues to be created and configured. The default queues for the supported data flows are listed below. While these queues would need to be created, the base product configuration already supports these data flows.

- Create Incident

  `OMS_TROUBLE_CALL`

- Get Customer Outage Status

  `OMS_CUST_STATUS`

  `OMS_CUST_STATUS_REPLY`

- Get Customer Outage History

  `OMS_CUST_HISTORY`

  `OMS_CUST_HISTORY_REPLY`

- Condition Updates and Queries

  `OMS_CONDITION_DATA`

- Outage Status

  `OMS_TROUBLE_CALL_STATUS`

- Customer Updates and Queries

  `OMS_CUSTOMER_DATA`

  `OMS_CUSTOMER_DATA_REPLY`

- SQL Transactions

  `OMS_SQL`

  `OMS_SQL_REPLY`

- Adapter Status Check

  `OMS_GATEWAY_STATUS`
- SQL Query

  `OMS_EXECUTE_QUERY`

  `OMS_EXECUTE_QUERY_REPLY`
- Customer Disconnect / Reconnect

  `OMS_CUSTOMER_DISCONNECT`

  `OMS_CUSTOMER_DISCONNECT_REPLY`
- Crew Outage Status Changes

  `OMS_TROUBLE_CALL_STATUS`
- Crew Updates / Crew Requests

  `OMS_UPDATE_CREW`

  `OMS_UPDATE_CREW_REPLY`
- Published Crew Updates

  `OMS_TROUBLE_CALL_STATUS`
- Network Trace

  `OMS_TRACE_NETWORK`

  `OMS_TRACE_NETWORK_REPLY`
- Area Summary

  `OMS_AREA_SUMMARY`

  `OMS_AREA_SUMMARY_REPLY`
- Callback List

  `OMS_TROUBLE_CALL_STATUS`

Table Used to Define MQ Queues

The following is the table definition for the MQ_ADAPTER_CONFIG table that defines the queues that will be used by the adapter. This is a configuration table that is loaded at startup.

Column	Type	Description
name	VARCHAR2(32)	Name of queue put queue for the reply queue, and for async output queues.
Q_type	INTEGER	Queue type: 1=async input, 2=async output, 3=request/reply
topic	VARCHAR2(32)	Subject matter for queue, specified in terms of document types
topic_type	VARCHAR2(32)	Message types for filtering specific document types.
translation	VARCHAR2(128)	Name of XSL translation to be applied, default=NULL
reply	VARCHAR2(32)	Name of reply queue (needed when qtype=3), but will be overridden if reply queue is specified on a request message
total_threads	INTEGER	Number of threads processing messages. Note: for async output queues the total_number is always set to 1.

The specific information topics to be supported include the following:

- PostError_001
- PostSrsOutput_001
- GetCustomerHistory_001
- GetCustomerStatus_001
- CreateIncident_001
- ExecuteQuery_001
- ExecuteTransaction_001

XSL Transformation Files

If XSL transformations are to be used by the interface, these files must be placed in a file directory on the adapter server. The directory path must be supplied on the adapter command line.

Default CES_GET and CES_PUT queues

CES_PUT and CES_GET queues are the default queues for sending and receiving each kind of Generic WebSphere MQ Adapter message. Both queues needed to be created, but they can be renamed through command line options defined above.

Trigger of broadcasting messages

Among information flows discussed in section 4, outage status flow <PostSrsOutput_001> and crew outage status change flow <PostSrsOutputStatus_001> are two broadcasting flows triggered by some actions happened in the Oracle Utilities Network Management System. Generally speaking, <PostSrsOutput_001> message will be generated every time when an internal srsoutput message is broadcasted. <PostSrsOutputStatus_001> message will be generated every time when an internal crew message is broadcasted. But the adapter configuration can selectively broadcast those messages based on the value in the mq_gateway_config.topic_type column which are internal srsoutput message and crew message type values.

Because Oracle Utilities Network Management System is highly configurable, it is pretty hard to define what kind operation to trigger internal srsoutput and crew messages and it is out of the scope of this document. Please reference corresponding SRS document for details of internal srsoutput and crew message.

Chapter 7

Generic WebSphere MQ Mobile Adapter

Introduction

This document describes the Generic WebSphere MQ Mobile Adapter that can be used by Oracle Utilities Network Management System customers to exchange data with external mobile data systems using MQSeries messages formatted using XML. The reader is assumed to have a working-level knowledge of Oracle Utilities Network Management System mobile data systems, XML, and MQSeries technologies.

Overview Description

Integration of Oracle Utilities Network Management System to a mobile data system involves the implementation of an adapter process, which exchanges messages with a mobile data system (MDS). The contents of messages sent to the MDS are generated from data obtained from the Oracle Utilities Network Management System services, transformed so that they are suitable for the MDS, and formatted into XML. The contents of messages received from the MDS are extracted from XML, transformed so that they are suitable for Oracle Utilities Network Management System, and sent to the Oracle Utilities Network Management System services.

The current implementation of the adapter supports the exchange of the following types of messages:

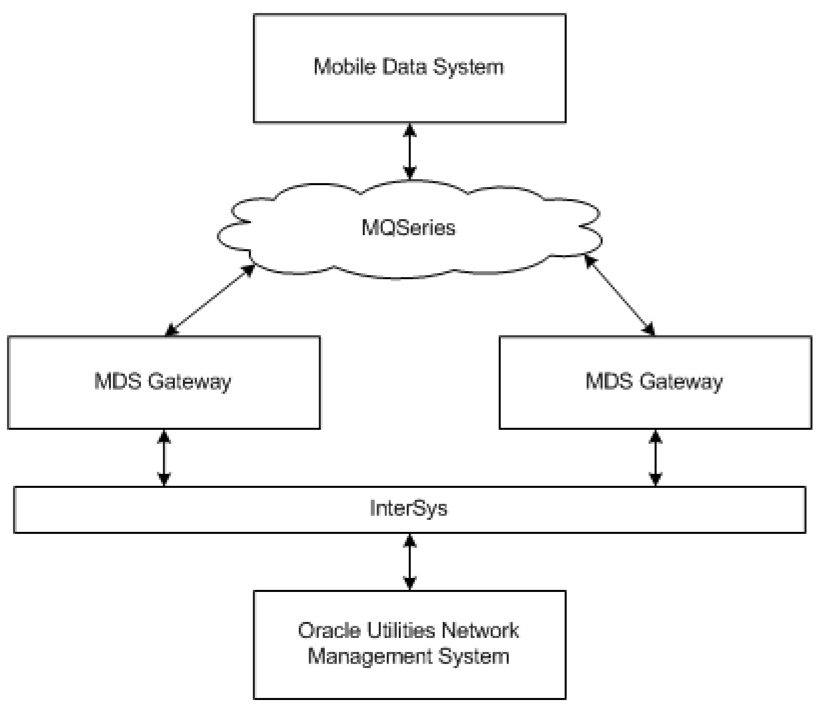
- Messages sent to the MDS
    - MDS order creation, update and cancellation messages, triggered by Event updates from Oracle Utilities Network Management System.
    - Interface communication status verification messages, triggered on a periodic basis.
- Messages received from the MDS
    - MDS order update and completion messages, triggering updates to the corresponding Oracle Utilities Network Management System events.
    - Crew log on, update and log off messages, triggering Oracle Utilities Network Management System crew creation, update and de-activation.
    - Crew assignment creation, update and deletion messages, triggering status changes to the corresponding Oracle Utilities Network Management System crews and events.
    - Interface communication status verification messages, triggering changes to the interface status.

Terms

InterSys	The middleware infrastructure which supports Oracle Utilities Network Management System
MDS	A Mobile Data System, which exchanges data with field crews' mobile data terminals.
MDS Order	A document or unit of work on the MDS. Also known under different names, for example Job
Event	Oracle Utilities Network Management System outage and non-outage events are generated by SRS based on customer calls and changes to the Oracle Utilities Network Management System topology model.
SRS	Service Reliability System, the service within InterSys which analyzes and manages outages
MQSeries	A queue-based messaging system developed by IBM.
XML	Extensible Markup Language
DML	Dynamic Message Language. This language is used to configure the adapter.

Functional Description

The purpose of this section is to describe the basic functional capabilities of the Generic WebSphere MQ Mobile Adapter, as applied to the integration of Oracle Utilities Network Management System with mobile data systems (MDS). While a high-level graphical description is provided here, detailed descriptions are provided in subsequent sections.



Functional Requirements

The key requirements for the current implementation of the Generic WebSphere MQ Mobile Adapter are:

- Generation of MDS orders based on Oracle Utilities Network Management System events.
- Updates of MDS orders based on changes to Oracle Utilities Network Management System events.
- Updates to Oracle Utilities Network Management System events based on updates to MDS orders.
- Creation and update of crew information based on changes to crews on the MDS.
- The ability to map multiple events to a single MDS order, based on event relationships in Oracle Utilities Network Management System. This allows groups of events to be viewed as single units of work on the MDS. An example of this is multiple events involved in a set of partial restoration steps.
- Support for multiple adapters in a High Availability configuration.
- The capability to send individual field changes to an order when events change so that there is a minimum usage of the limited bandwidth available to transmit data to and from field crews. This requires storing the data last sent to the MDS so that change detection can be used. This data is saved to the database, so that change detection can be used over an adapter shutdown and restart.
- The ability of data from various Oracle Utilities Network Management System sources to be transformed and combined for transmission to the MDS. The data sources include the Oracle Utilities Network Management System services and the database. Data from the Oracle Utilities Network Management System services will be obtained from both asynchronous notification messages and the use of API calls.
- The ability of data from the MDS to be transformed and sent to various Oracle Utilities Network Management System destinations, using database updates and API calls.

Hardware and Software Requirements

The purpose of this section is to describe the environment relevant to this interface.

Oracle Utilities Network Management System Environment

The Oracle Utilities Network Management System environment consists of a number of servers that are interconnected using the InterSys messaging system.

Adapter Server

The Generic WebSphere MQ Adapter environment may be resident on the same servers as the Oracle Utilities Network Management System services, or it may be implemented on a separate server. Specifications for a stand-alone adapter server:

- All Oracle Utilities Network Management System UNIX operating systems are supported
- IBM WebSphere MQ 6 messaging product (Note, however, that the queues may reside on a remote machine) must be installed.
- A LAN connection to the Oracle Utilities Network Management System server must be available
- ISIS must be installed and configured

Depending upon the high availability scheme selected, it would be possible to configure more than one adapter server for redundancy.

Oracle Utilities Network Management System Server

The Oracle Utilities Network Management System server environment is typically deployed on one or more UNIX servers configured with the following:

- UNIX operating system
- Oracle RDBMS with Oracle Utilities Network Management System model
- Oracle Utilities Network Management System service processes
- LAN connection to adapter server
- Message queues to be used by the MQ/XML Adapter appropriately declared in the defined database configuration table.
- ISIS

External System Environment

The external system is any system that can exchange information with Oracle Utilities Network Management System through an adapter. The environment of the external system has the following capabilities:

- Any operating system which supports IBM WebSphere MQ messaging
- IBM WebSphere MQ messaging product
- Applications that can request or publish information in a manner which is either directly or indirectly (through a translator) compliant with the XML specifications contained within this document via queues
- Queues must be pre-configured
- IBM WebSphere MQ Integrator can be used as needed for routing and translation.

Required Installed Software

The following lists the required software that needs to be installed prior to any configuration of the Oracle Utilities Network Management System Generic WebSphere MQ Adapter.

IBM's WebSphere MQ Version 6

ISIS (installed as part of the base Oracle Utilities Network Management System installation)

> **Note**: ISIS is the messaging backbone for Oracle Utilities Network Management System and will already be present on any Network Management System servers. If the Generic WebSphere MQ Adapter is to be executed on a separate server than the Network Management System, then that server must also have ISIS installed and running. Every server installation must be running the same version of ISIS. The CMM_CELL environment variable must be set the same on any servers which are to communicate through ISIS.

Adapter Installation

Overview

This section is used to guide the user in the installation of the Oracle Utilities Network Management System Generic WebSphere MQ Adapter. The following are assumed to be true before the adapter is installed:

1. Oracle Utilities Network Management System is installed and functional. This means that database access has been confirmed, as well as ISIS message bus communication.
2. WebSphere MQ is installed on a machine that is accessible to the Oracle Utilities Network Management System.

Check if the Generic WebSphere MQ Mobile Adapter is installed

Verify that the following files are found in their respective folders

- $CES_HOME/bin/mdsgateway
- $CES_HOME/bin/ces_mds_gateway.ces

Configure Adapter to run as NMS System Service

Configure the Generic WebSphere MQ Mobile Adapter to run as an Oracle Utilities Network Management System service by updating the `$NMS_HOME/etc/system.dat` file to include the Generic WebSphere MQ Mobile Adapter as a system service. There are 3 main sections where this service needs to be defined: the service, program and instance sections.

See the `$CES_HOME/templates/system.dat.template` file for examples of how to configure the Generic WebSphere MQ Adapter. Search for "`mdsgateway`" in the file and copy those lines to `$NMS_HOME/etc/system.dat` file. Make sure all lines are uncommented so that they are active.

See the command line options section below for more details on available options. You must restart the system services in order to the Generic WebSphere MQ Mobile Adapter to properly be monitored by SMService.

> **Note:** In setting up `$NMS_HOME/etc/system.`dat, it is important to note that the examples above were presented only for illustration purposes. Parameters may differ on an actual project setting. Coordinate with your Project Engineer in setting up your system configuration file. Also, take note that in the example above, it is assumed that the Generic WebSphere MQ Mobile Adapter will reside on the same machine where the Oracle Utilities Network Management System environment resides.

Generic WebSphere MQ Mobile Adapter Command Line Options

The command line for the Generic WebSphere MQ Mobile Adapter provides the following options:

- **-install:** Install the adapter as a Service.
- **-uninstall:** Un-install the adapter as a Service.
- **-override:** Run the adapter as a normal application, not a Service.
- **-debug <level>:** Output debug messages to the log from the adapter and all API toolkits.
- **-dl:** Output level zero (fewest) debug messages from the adapter, only, to the log. (Note that this is the letter 'l').

- **-d1**: Output level one and lower debug messages from the adapter, only, to the log. (Note that this is the number '1').
- **-d2:** Output level two and lower debug messages from the adapter, only, to the log.
- **-d3:** Output level three and lower debug messages from the adapter, only, to the log.
- **-logfile <file name>:** The name of the file to output error, warning and debug messages to. If no file name is specified, MDSlog.txt in the folder the adapter is started in is used. If this option is not present, messages are sent to standard output. If the adapter is being run as a Service, it is highly recommended that the full pathname of the log file be provided. The format of the full path name uses double backslashes (\\) rather that a single backslash (\). For example: c:\\mdsg\\ Mdslog.txt.
- **-relog <relog time in hours>:** The relog period in hours. The log file continues to grow as the adapter runs. This can potentially fill up a disk. To avoid this, the adapter has the facility to close the log file, save it in the logs directory and open a new log file. This can be achieved using the "relog" high level message, and/or by specifying a relog time in this command line option. The default is 24 hours. Specifying a relog period of zero disables periodic relogging.
- **-dbserver <database server name>:** Specify the database server to use. The adapter can be a heavy user the database. To prevent it from having an impact on other users and to prevent other users from having an impact on it, it can be configured to use its own database server. The default is to use the normal DBService. To configure the use of its own database server, use this command line option, and make sure that an instance of DBService is running with the database server name as its –service command line option.
- **-waitfor <service name>:** The name of a service to wait for before beginning initialization. It is highly recommended that the adapter waits for the database server it uses. The default is DBService. If the –dbserver option is used, this option should be set to the name that the instance of DBService has been given in its process_name option. Specifying the empty string as the name disables this feature
- **-maxwaitfor <time in seconds>:** The maximum time to wait for the service the adapter waits for before beginning initialization. If the service does not respond within this time, the adapter exits with a fatal error message. The default is 120 (two minutes).
- **-dmldir <directory path>:** The path of the directory that the configuration files included by the configuration files at the end of the command line. If the adapter is being run as a Service the full path of the directory should be specified, using forward slashes (/) rather than backslashes(\). For example: c:/mdsg/data.

Any other command line arguments are assumed to be file names of configuration files to use. They are processed in the order that they appear in the command line. If the adapter is being run as a Service, the full path name of the files should be specified.

Optionally Configure the Adapter to Run with another Instance of DBService

In `$NMS_HOME/etc/system.dat`, include the MQDBService as one of the services. Use the TCDBService entries as examples of how to set this up.

> **Note**: If using a separate DBService, you must start the Generic WebSphere MQ Mobile Adapter with the "-custdbsname" command line parameter and use the MQDBService name as the argument.

Configure the WebSphere MQ Server

References to "Console Root" throughout this write-up refer to the highest level in the tree displayed by the WebSphere MQ Explorer GUI.

Create New Queue Manager

From WebSphere MQ Explorer tree, select:

- Console Root ==> WebSphere MQ ==> Queue Managers ==> New ==> Queue Manager
- Queue Manager (name) = NMS_MGR.A
- Check Make this the default Queue Manager (indicating yes)
- Click Next – use default settings (circular logging)
- Click Next – use default settings (start queue manager)
- Click Next – uncheck "create listener configured for TCP/IP"

Create New Queues (2)

From WebSphere MQ Explorer tree, select:

- Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==>Queues ==> New ==> Local Queue
- Queue Name = NMS.A.FROMNMS
- Click **OK** – use all default settings
- Click Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Queues ==> New ==> Local Queue
- Queue Name = NMS.A.TONMS
- Click OK – use all default settings

  Note: At this point, the two new queues should be created. Check the status of each queue or put a test message into each queue by doing the following:

- Select Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Queues

  This should display a list of queues.

- Right click on the desired queue to bring up a menu containing selections for Status and Put test message.

Create Server Connection Channel

- From WebSphere MQ Explorer tree, select
- Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Advanced ==> Channels ==> New ==> Server Connection Channel

  A dialog will display containing tabs for General, Extended, MCA, Exits, and SSL

- In the General tab, the Channel Name is SCH1
- In the MCA tab, the MCA User ID is the local login userid
- Click OK – use all default settings

  Note: At this point, the new server connection channel should be created. Check the status of the new server connection channel by doing the following:

- Select Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Advanced ==> Channels.

  This should display a list of connection channels.

- Right click on the SCH1 channel to bring up a menu containing selections for Status and Start and Stop.

- Select Start. The new server connection channel should display the message "The request to start the channel was accepted (amq4008)".

Create New Queue Manager Listener Service

The default TCP/IP port for the default Queue Manager listener is 1414. Multiple listeners can be configured, but for simplicity, in this case, the original installation default listener for the default queue manager has been stopped. This frees up port 1414 for use by a new listener.

Stopping the original default Q manager listener

- Select Console Root ==> WebSphere MQ Services (local) ==> (*the original default queue manager name*).

This will cause a list of services to be displayed, one of which is the "listener" service.

- Right Click on "listener ==> properties" and stop the listener.
- Change the startup from Automatic to Manual.

This listener should no longer start-up at reboot.

Create new Q manager listener for new Q manager

- Select Console Root ==> WebSphere MQ Services (local) ==> NMS_MGR.A ==> New ==> Listener.

This will invoke a dialog to create a new "listener" service. This dialog will have three tabs, General, Recovery, and Parameters.

- The Parameters tab port number must be 1414.
- The General tab startup type should be "Automatic".
- Then press the "start" button on the General tab.
- To check the status of the listener, select Console Root ==> WebSphere MQ Services (local) ==> NMS_MGR.A ==> Listener ==> Properties

Configure the MQ Client

Set environment variables

- The environment configuration file (nms.rc), which is a data file listing Oracle Utilities Network Management System environment settings, should have the following:

  ```
  export MQSERVER=SCH1/TCP/10.115.3.85
  ```

- The environment configuration file must also have two variables set to locate the .TAB file for WebSphere MQ. The .TAB must be copied to the MQ client from the MQ server host as specified by these variables. Examples:

  ```
  export MQCHLLIB=/users/proj/MQ
  export MQCHLTAB=AMQCLCHL.TAB
  ```

- To review IBM's documentation on the MQSERVER environment variable, click on the following URL:

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzal.doc/csqzal10203.htm

- View this environment variable (to ensure that it's correct) by typing in the following command: echo $MQSERVER

Test the connection between MQ Client and MQ Server

Test the server connection channel (amqscnxc)

- On the Unix command line, type in the following command:

  ```
  /usr/mqm/samp/bin/amqscnxc -x 10.115.3.85 -c SCH1 NMS_MGR.A
  ```

  where:

  - -x is the IP address of the MQ Server host
  - -c is the Server Connection Channel Name
  - the third parameter is the desired Queue Manager Name

Test 'putting' a message from Server to Client (amqsputc)

- On the Unix command line, type in the following command:

  ```
  /usr/mqm/samp/bin/amqsputc NMS.A.FROMNMS Sample AMQSPUT0 start
  target queue is NMS.A.FROMNMS

  <MSG-FROM-SVR>VOILA</MSG-FROM-SVR> Sample AMQSPUT0 end
  ```

- The message should appear in the queue named NMS.A.FROMNMS which can be viewed on the client using the MQ Explorer GUI at:

  ```
  Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==>
  Queues ==> NMS.A.FROMNMS ==> Browse Messages
  ```

Test 'getting' a message on Client from Server (amqsgetc)

- First "get" the message just written

```
/usr/mqm/samp/bin/amqsgetc NMS.A.FROMNMS Sample AMQSGET0 start message
<<MSG-FROM-SVR>VOILA</MSG-FROM-SVR>> no more messages Sample AMQSGET0
end
```

Test 'putting' a message from Client to Server using WebSphere MQ GUI

- Select Console Root ==> WebSphere MQ ==> Queue Managers ==> NMS_MGR.A ==> Queues ==> NMS.A.TONMS ==> Put Test Message
- Paste the following into "Message Data":
- <MSG-FROM-CLNT>VOILA</MSG-FROM-CLNT>
- Click OK.

  The following message should be displayed: "The test message was put successfully (amq4016)".

Test 'getting' a message on Server from Client (amqsgetc)

On the Unix command line:

```
/usr/mqm/samp/bin/amqsgetc NMS.A.TONMS Sample AMQSGET0 start message
<<MSG-FROM-CLNT>VOILA</MSG-FROM-CLNT>> no more messages Sample
AMQSGET0 end
```

Configure Queues for Required Data Flows

All incoming (with respect to NMS) data flows would go through the NMS.A.TONMS queue. All outgoing messages would go through the NMS.A.FROMNMS queue. Additionally, an error queue is intended for all adapter generated error messges.

These queues must be added to the DML configuration with the following parameters:

```
config_QueueManager_name
config_OutQueue_<id>_name
config_InQueue_<id>_name
```

where <*id*> is a unique identifier to differentiate between multiple output or input queues.

Design Overview

The adapter passes data between Oracle Utilities Network Management System and the MDS, transforming the data based on the configuration files and tables. Messages sent and received using MQSeries are formatted in XML. The types of messaging that are supported include the following:

- Asynchronous publish from Oracle Utilities Network Management System to MDS (using 'fire and forget' pattern)
- Asynchronous publish from MDS to Oracle Utilities Network Management System (using 'fire and forget' pattern)
- Request/reply from Oracle Utilities Network Management System to MDS (the requestor can process this either synchronously or asynchronously)
- Request/reply from MDS to Oracle Utilities Network Management System (the requestor can process this either synchronously or asynchronously)

There are a number of mobile data systems produced by various vendors. These systems allow the end customer to specify the contents of orders that are sent to the crews and the various ways that the crews can report on their progress of their work in the field. In addition, crews can request information from the dispatcher and from other systems in the customer's organization. The number of options and capabilities vary between the mobile data system vendors.

Oracle Utilities Network Management System allows the end customer a large number of options in the kinds of data that are associated with different kinds of objects, including customers, customer calls, outage and non-outage events, field crews, and devices in the network model.

This implies that an adapter that interfaces between Oracle Utilities Network Management System and multiple mobile data systems and multiple implementations of Oracle Utilities Network Management System and the various individual mobile data systems has to be highly configurable.

To meet this challenge, the Oracle Utilities Network Management System WebSphere MQ Mobile Adapter can be configured to perform data manipulations and business logic, for flexibility, and has functions commonly used interfacing between Oracle Utilities Network Management System and mobile data systems hard coded for performance.

Configuration Concepts

The core of the configuration is one or more file(s) written in Dynamic Message Language (dml). This language allows the adapter to dynamically generate XML messages from various data sources, and to process XML messages to distribute the data contained in the message to various data sinks.

DML allows data to be textually transformed and combined, has logic to allow XML elements to be included or ignored, and different InterSys API calls to be made, all depending on the data being processed.

The basic units in dml are known as documents. There are two main types of document:

- Output Documents: These instruct the adapter how to generate XML documents to be sent to the MDS. For example, an Output Document is used to generate order create and update messages.
- Input Documents: These instruct the adapter how to process the data received in XML documents from the MDS. For example, an Input Document instructs the interface how to process a crew creation message.

One or more dml files are read by the adapter during initialization. The contents of these files are compiled into a hierarchical set of internal data structures. As the various documents are triggered, the internal data structures are used to generate outgoing XML messages and process incoming XML documents.

The details of the syntax and capabilities of dml are described below in paragraph **DML Files** on page 7-15 and in the Appendices.

Output Documents

An Output Document gathers data from various sources, performs all necessary data transformations and logic, and formats the resulting data into XML elements whose tags are specified in the Output Document. The main sources of data are asynchronous InterSys messages, database selections, data cached in the adapter from previous XML documents, both input and output, and InterSys API calls.

In order to have the resulting XML document generated, the Output Document has to be activated. This activation is known as “triggering” Output Documents. These triggers occur under the following circumstances:

- The reception of an InterSys message. For example, an update to an event from SRS could trigger an order update message.
- Periodically. For example, an interface status message that produces a reply from the MDS to monitor the state of the connection between the two systems.
- By request in another document. For example, an assignment message is received from the MDS, but Oracle Utilities Network Management System has no record of the specified crew. In this case the Input Document processing the assignment message could trigger a query to the MDS, asking it to return the data describing the crew.
- When event relationships change. For example when a number of events are grouped manually on the Work Agenda.

Input Documents

Input Documents process data from elements whose tags are specified in the Input Document. Other elements are ignored. Once the data from all tags in the message that the Input Document recognizes are gathered, all necessary data transformations are performed, possibly in conjunction with data from other sources, and the results are passed to various data sinks. The main data sinks are the database, InterSys API calls, and the adapter cache for use in later document processing.

In order for Input Documents to process the appropriate incoming XML documents, Input Documents have selection criteria. These criteria specify one or more of the following conditions that have to be met before the Input Document is used to process the data in the XML document:

- The tag and attributes of the XML document's root element.
- One or more element tags that are required to be in the XML document to identify its usage. Optional elements can be processed with the required elements. Facilities are available to set optional elements to default values, or to alter the processing logic depending whether they are present or not.
- The queue that the XML document was received on.

Integration with System Services

In order for the various dml documents to access Oracle Utilities Network Management System service objects, a number of facilities are available. They include:

- Asynchronous notification messages. For example, SRSoutput messages that describe new or modified events. DML documents can access the data supplied by these messages.
- Access to the database. DML documents can read data from the database using select statements, and can write data using insert and update statements.
- Access to the service APIs. Service APIs relevant to objects used in interfacing to mobile data systems are available to dml documents using function calls.

Aggregation of Objects

Oracle Utilities Network Management System and mobile data systems perform very distinct functions and, therefore have distinct views of a utility. This leads to different object models. The difference that has the major impact on the integration of the two is what appears to be the same object on each side of the interface: events and orders.

Events are associated with particular devices in the electrical network, while orders describe work to be done in the field. Normally, one event is associated with one order, for example when a customer transformer needs to be replaced because of a fault. However, in more complex situations, this one-to-one relationship no longer applies.

An example of this is the series of steps involved in a partial restoration. A typical scenario is presented below:

1. A fault occurs in an underground loop, causing the fuse protecting the loop to blow.
2. This results in an outage event on the fuse.
3. This generates an order, which is sent to a crew in the field.
4. The crew arrives on site and discovers that an underground cable has been cut at the end of the loop opposite from the fuse.
5. In order to restore power to as many customers as possible, the crew opens the switch closest to the cut cable, and then replaces the fuse.
6. This creates a new outage event associated with the downstream switch.

7. The event associated with the fuse remains, but is now in a restored state.

If a one-to-one relationship were maintained between orders and events, there would now be a new order associated with the downstream switch. However, this does not match the view of the crew in the field. The new event is merely part of the work involved in servicing the original order.

To accommodate this, the adapter will aggregate partial restoration events, if the appropriate configuration options are chosen, into what appears as a single event to the dml. The dml can then process the aggregate event into a single order. Similarly, when the dml processes updates to an order sent by the MDS, causing updates to the aggregate event, the adapter applies the updates to all of the aggregated events in Oracle Utilities Network Management System.

Another example is when the Oracle Utilities Network Management System operator wants to group related events, for example when multiple outages occur in an ice storm in a small area, and assign them to a single crew. The adapter can be configured to treat this in the same way as a partial restoration, but this is not necessarily the preference of the customer. Some mobile data systems can group orders into a folder like object. The adapter can provide a trigger to the dml to process the group of associated events appropriately.

Information Flows

The contents of the configuration files are driven by the information flows required for a particular customer.

High Availability

The goal of the Oracle Utilities Network Management System WebSphere MQ Mobile Adapter redundancy is to provide assured message receipt and delivery between the Oracle Utilities Network Management System services and the MDS. This is achieved using Microsoft Windows 2000 Server Clustering with an appropriate IBM MQSeries configuration. Configuring Clustering and MQSeries is not addressed here.

The adapter supports High Availability in the following ways:

- Order data is saved to the database so that change detection can continue over a shutdown restart cycle.
- The use of the MQSeries syncpoint facilities to preserve input messages that have not been fully processed.

Performance

This interface is intended to provide for high performance needed to process frequent message exchange such as in the case of a high volume of events during a storm. In order to provide optimum performance, there are aspects of both implementation and usage. Aspects of usage include:

- Reducing the number of database accesses and API calls required to generate outgoing XML documents and to process incoming XML documents.
- Reducing the number of elements that need to be sent and received.
- Keeping the number of critical elements that cause XML documents to be sent when they change to as low a number as possible.
- Allowing events time to group for as long as possible, before they cause the creation of orders and cancellation of orders for grouped calls. An example of this is to wait until an event is acknowledged before creating the order, and implementing business practices that delays the acknowledgement until the event is ready to be processed by the MDS.
- Sending multiple requests and updates from the MDS together in a single message.

Aspects of implementation which optimize performance include:

The assumption that incoming XML messages are well formed, bypassing the validation step. It is assumed that the sender provided well-formed XML, which was transmitted using reliable communication mechanisms. The actual validation test is whether or not the code that internally parses a message can extract a sufficient set of parameters to process an Input Document. XML that is not well formed will typically generate an error. It should also be noted that XML validation does not necessarily guarantee valid information provided by an external system. If this generates an error in the adapter, the error will be logged.

High Level Messages

High-level messages are typically used within Oracle Utilities Network Management System to permit one process to control another process. There are no special high-level messages that would be required for the Generic WebSphere MQ Mobile Adapter.

Note that doing an Action any.any stop will stop the adapter, which needs to be taken into consideration for administering the adapter when starting and stopping it.

Supported high level messages include the following:

- debug <debug level>: set the global debug level to <debug level>. If no level is supplied, toggle between 0 and 1.
- dl <arg>: modify the local mdsadapter debug level depending on arg
  - (none): toggle between 0 and 1
  - off: turn off debug (no messages)
  - on: set to 0 (lowest level above off, least messages)
  - (a number): set to number (the higher the more debug messages)
- dump: dump adapter data to the log
- isisdump: request an isis dump
- report
- stop
- relog: close the current log file, save it in the logs directory, and open a new log file
- trigger <document name> <trigger name> [<arg> …]: Trigger the specified OnRequest trigger in the specified Output Document passing all the additional arguments to the document. If the document name, trigger name, or the number of arguments is invalid, the adapter exits due to a configuration error.

Information Models

The key objects supported by the adapter include:

- Incidents are typically related to a customer and are generated by trouble calls. The customer in turn is related to a device. In the absence of a correlation to a device, a trouble call is classified as a ‘fuzzy’ call, which differentiates it from a call that can be directly correlated to the electrical distribution network.
- Events are a consequence of the correlation of incidents. Outages are one form of an event that is managed by SRSService. Some events are non-outage events, such as power quality. The type of call that is provided can identify such non-outage and outage events. Each call needs to be identified with a trouble code, which will determine the type of call that SRSService will generate within Oracle Utilities Network Management System.

- Devices, which are part of the electrical distribution network. Customers, outages and conditions may have relationships to devices. Typically customers are related to transformer devices. Outages are typically related to switch, fuse or transformer devices.
- Crews, who work in the field that can be made up of one or more crew members, and one or more vehicle.
- MDS orders, which contain data relevant to the work that the crews perform in the field.

Configuration

There are several mechanisms used to configure this interface:

- DML files
- Database tables
- Command line options

Once the dml files and configuration tables for a customer's initial configuration have gone into production, a knowledgeable user can make changes to the configuration, as business needs change. For example:

- Change element tags and attribute names in input and output messages.
- Remove obsolete elements and attributes.
- Add new elements and attributes.
- Change the format and contents of elements and attributes in output messages.
- Change the transformation of data in input messages.
- Alter business logic.
- Change the contents of the tables that translate Oracle Utilities Network Management System values to and from equivalent values in the messages to and from the MDS.
- Change the conditions that trigger the various messages sent to the MDS.
- Alter the names of the MQSeries queue manager and queues.

DML Files

DML files contain dml code that is compiled into internal data structures during initialization. Any errors in the dml files are logged, with the file names and line numbers that the errors were detected on. Any such errors cause initialization to fail. There is an off-line program that allows dml files to be checked before use.

A narrative about the various facilities is provided here. The details of the syntax and capabilities of dml are described below in the Appendices.

The main purpose of dml is to generate and process XML elements. The first few sections describe the generation of XML from Output Documents, but most of the facilities described can be used in Input Documents, which process XML. The discussion of Input Documents starts at section **Input Elements and Attributes** on page 7-19.

Output Elements and Attributes

The following is an example of a dml statement that will generate an element:

```
&Hello = world;
```

This will generate the element:

```
<Hello>world</Hello>
```

The ‘&’ identifies the following string ‘Hello’ as an element tag. The ‘=’ assigns the element’s data. The ‘world’ is a constant value. The ‘;’ terminates the statement.

Similarly, the statement:

```
&Tag < attr1=1; attr2=two; > = "";
```

generates the following empty element with two attributes:

```
<Tag attr1="1" attr2="two"/>
```

In the statement the ‘<’ ‘>’ pair enclose a list of attributes and their values. The "" is the empty string.

The statements:

```
&Tag < attr1=1; attr2=two; >
{
    &SubTag1=SubData1;
    &SubTag2=SubData2;
}
```

generate the following element with two attributes, and two sub-elements:

```
<Tag attr1="1" attr2="two">
<SubTab1>SubData1</SubTab1>
<SubTab2>SubData2</SubTab2>
</Tag>
```

The ‘{‘ and ‘}’ enclose a list of sub-elements.

External Data

So far, the elements and attributes discussed have contained only constant data. An example reading data from an event in an SRSoutput message is:

```
&EventDevice=$E.devAlias;
```

This assigns the alias (name) of the event’s interrupting device to the element tagged ‘EventDevice’. The ‘$’ indicates that the following string is the name of an external data object. The ‘E’ indicates that the data is associated with an event object, the ‘.’ separates the components of the name, and the ‘devAlias’ identifies the data field within the object. The names used for the fields of an event object are similar to the names of the corresponding fields in the Oracle Utilities Network Management System SRSoutput class.

Data obtained this way can be combined with other data using dml operators. For example:

```
&ExternalId = "NMS:" + $E.outageHdl.idx;
```

concatenates the constant string ‘NMS:’ (contained within quotes because of the non-alphanumeric character ‘:’), with the event’s index (sometimes known as the Ticket Number). Assuming that the event’s index was 1234, the following element would be generated:

```
<ExternalId>NMS:1234</ExternalId>
```

Variables

Sometimes the same data is to be assigned to more than one element, or intermediate result is to be used in more than one element. Variables are used to hold such intermediate results. For example:

```
@eid = "NMS:" + $E.outageHdl.idx;
&ExternalId = @eid;
&code = @eid + "%" + $E.devAlias;
```

would generate, assuming the device name was XFM1234567,

```
<ExternalId>NMS:1234</ExternalId>
<code>NMS:1234%XFM1234567</code>
```

The ‘@’ indicates that the following string is the name of a variable. The ‘+’ operator concatenates the string to the right with the string to the left.

SQL Select Statements

A SQL Select statement is used to read data from the database. For example, the statement:

```
sqlselect @type, @intDev | type_om, interrupt_dev_om |
          picklist_info_upd_tr | "where ref_id=" | $E.outageHdl.idx;
```

sets variables ‘type’ and ‘intDev’ to the values in the ‘type_om’ and ‘interrupt_dev_om’ columns of the row in the ‘picklist_info_upd_tr’ table where the ‘ref_id’ column matches the current event’s index.

The ‘sqlselect’ introduces the SQL select statement, and is followed by a list of variables that will be set to the contents of the columns in the database, separated by commas, and terminated by a vertical bar (|) as a separator. After the variables is a list of the columns to assign to the respective variables, separated by commas, and terminated by a vertical bar. The number of column names must match the number of variables. The column names are followed by the table or view name. The rest of the statement builds the ‘where clause’, by concatenating the values of all the remaining components. This syntax only supports a simple select syntax. More complicated select statements are built as a view on database tables, which the dml then accesses using this simpler syntax.

The table name can be a constant or the value of a variable, an external data field, a function call, or the result of the data manipulations, as described below.

If the table name is a constant, the columns are checked for validity at initialization time. If any of the columns specified are not in the table, an error message is output to the log, and initialization fails.

If the table name is not a constant, the columns are checked for validity at run time. If the table does not exist, all variables are set to their default values (described below), and, optionally, a debug message is output to the log. If a column is missing, the variable for the column is set to its default value, and, optionally, a debug message output to the log.

An important source of data relating to devices in the model is the facilities (or attribute) tables associated with various device classes (types). The name of the table for a device class is supplied by the ‘classTable’ function, which queries the Oracle Utilities Network Management System services for the table’s name. Not all facilities tables have the same columns (for example fuses and transformers may have fuse sizes, while reclosers may not). This is the reason that the adapter does not regard a missing column at run time as a fatal error.

Some tables, for example the facilities tables described above, contain data that does not change very often, and can be regarded as remaining static. In this case, values only need to be read from these tables when the table name, or ‘where clause’ changes. Such tables are declared static by use of the ‘`static`’ keyword. For example:

```
sqlselect @devPhases,        @fuseSize |
          phase_designation, fuse_size |
static classTable($E.devHdl.cls) | "where h_idx = " | $E.devHdl.idx;
```

reads the ‘phase_designation’ and ‘fuse_size’ columns for the interrupting device for an outage.

If the select returns no rows, the variables are set to their default values. If the select statement returns one or more rows of the table, the values are taken from the first row returned. Subsequent rows are ignored in the initial implementation of the adapter.

Function Calls

Function calls have a number of uses, including:

- Making Oracle Utilities Network Management System API calls.
- Performing data manipulations not supported by the dml syntax.
- Accessing the adapter's data.
- Accessing other configuration data.

A simple example of truncating external data to 100 characters to match the length of strings expected by the MDS is:

```
&Address = substring($E.dispAddress, 0, 100);
```

the Address element is assigned 100 characters of the 'dispAddress' field from SRSoutput, starting at the beginning of the string (an offset of zero).

The various functions available are described in **DML Function Calls** on page 7-70.

Expressions

The values of the various data sources can be combined in expressions. Examples of the string concatenation operator '+' have been shown above. The other operators in the initial implementation involve boolean or logical values.

Boolean values are either true or false. Any non-empty string is considered true, while the empty string is false. Boolean operators return 'Y' when they evaluate to true and the empty string when false. The boolean operators are:

- The equality operator '`==`': This compares two values for textual equality. For example `abc == ab` has a value of `""` (false).
- The inequality operator '`!=`': This compares two values for textual inequality. For example `abc != abc` has a value of `""` (false).
- The logical AND operator '`&&`': This is the union of two values, and has a value of true if and only if both values are true. For example, `Y && Y` has a value of Y (true).
- The logical inclusive OR operator '`||`': This is the intersection of two values, and has a value of true if either of the values is true. For example, `@v1 || Y` has a value of Y (true) whatever the value of the variable v1.
- The logical NOT operator '`!`': This inverts a single value, being true if the value is false, and vice versa. For example `!@v1` is exactly equivalent to `@v1 == ""`.

Boolean values can be used directly, when the Y or "" is appropriate, or in the alternation operator, described here, or by if statements, described below.

The alternation operator returns the value of one of two expressions, depending on the value of a boolean expression and takes the form:

```
@bool ? @vtrue : @vfalse
```

If '@bool' is true, the expression's value is that of '@vtrue', and if '@bool' is false, the value is that of '@vfalse'.

Operator Summary

In the table below, each operator is followed by its name and an example of its use.

Operator	Description	Example
!	Not	!@bool
+	Concatenate	"NMS:" + $E.outageHdl.idx
==	Equal	@v1 == @v2
!=	Not equal	@v4 != on
&&	Logical AND	@b1 && @b2
||	Logical inclusive OR	@b3 || @b4
? :	Alternation	`@bool ? @vtrue : @vfalse`

The not operator is right associative, all the others are left associative. For example, `!!@v` means `!(!@v)`, and `@v1 + @v2 + @v3` means `(@v1 + @v2) + @v3`, and `@v1 == @v2 == @v3` means `(@v1 == @v2) == @v3`.

Each box holds operators with the same precedence. An operator has a higher precedence than those in lower boxes. For example, `@v1 == @v2 + @v3` means `@v1 == (@v2 + @v3)`.

Parentheses can be used to change the precedence. For example, `(@v1 == @v2) + abc` would result in "Yabc" if @v1 was the same as @v2, otherwise "abc", while `@v1 == @v2 + abc` would result in "Y" or "" depending on whether the value of v1 is the same as the value of v2 concatenated with "abc".

Parentheses should be used whenever the precedence is in doubt, especially when a boolean expression becomes more complex.

Input Elements and Attributes

The element statements described above in section 7.1 are used in Output Documents. Elements in Input Documents do not have values assigned to them, as the element values are supplied by the incoming XML. An example of an input element follows:

```
&JobNumber;
```

This defines an element whose tag is 'JobNumber'. This element is optional, i.e. it does not have to be present in the input XML for the enclosing Input Document to be processed. In addition, no attributes were declared for the element. Therefore any attributes for the element in the input XML will be ignored.

If the element is not present in the input XML, its value is the element's default. The specification of default values is described below. In this example, the default is "", the empty string. If special processing should take place when an element is present or not, the boolean function 'isSet' can be used to alter the processing logic using, for example, an 'if' statement, described below.

The value of the element can be used in assignments, expressions, and in the various SQL statements.

For example:

```
@jobNo = &JobNumber;
```

assigns the element's value to the variable 'jobNo'.

To declare that an element is required for the Input Document to be processed, the required flag 'R' would be added to the declaration, as described below in section **Flags** on page 7-23.

An example of an element with four sub-elements is:

```
&CrewKey
{
&CrewName;
&AgencyCode;
&ShiftCode;
&ShiftDate;
}
```

The default way to obtain the value of a sub-element is to specify parent's tag, followed by a forward slash ('/'), followed by the sub-elements tag. Deeper nesting of elements is similar. For example:

```
@crewId = &CrewKey/CrewName + &CrewKey/AgencyCode + &CrewKey/ShiftCode
+      &CrewKey/ShiftDate;
```

If the nesting is very deep, it can become very tedious and error prone to have to type all the tags leading to a particular element. This can be avoided by giving the element a name, as described in section **Alternate Names** on page 7-24.

Input attributes are declared similarly to Output Document element attributes. For example:

```
&Elem < attr1; attr2; >;
```

declares an element with a tag of 'Elem', with two attributes 'attr1', and 'attr2'.

The value of an attribute can be obtained as follows:

```
@val = &Elem<attr2>;
```

The attribute's name is enclosed in angle brackets '<' and '>'.

SQL Insert and Update Statements

SQL Insert and Update statements are used to save data to the database. For example, the statement:

```
sqlinsert picklist_completion_log |
       ref_id,        who,      reason_for_update,  when |
       $O.event.idx, @crewId, @reason_for_update, time();
```

inserts a row into the 'picklist_completion_log' table. The value of the 'ref_id' column is supplied by the index of the Handle held in the 'event' field of the external object 'O' (Order), that is associated with a particular MDS Order. The value of the 'who' and 'reason_for_update' columns are supplied by the variables 'crewId' and 'reason_for_update'. The 'when' column is supplied by the 'time' function, which returns the current time in internal format.

The 'sqlinsert' introduces the SQL insert statement, and is followed by the table or view name, followed by a vertical bar (|) as a separator. The list of column names follows, separated by commas, and terminated by a vertical bar. The list of data sources follow, separated by commas, and terminated by the statement terminator, ';'. The number of sources must match the number of columns.

The table name or a data source can be a constant or the value of a variable, an element, an external data field, a function call, or the result of an expression.

If the table name is constant, the columns are checked for validity at initialization time. If any of the columns specified are not in the table, an error message is output to the log, and initialization fails.

If the table name is not constant, the columns are checked for validity at run time. If the table does not exist, nothing is sent to the database, and, optionally, a debug message is output to the log. If a column is missing, the data for the column is not sent to the database, and, optionally, a debug message output to the log.

The type and length of the columns are read from the database, the first time a table is accessed (commonly at initialization time). Values for character columns (CHAR or VARCHAR2) are truncated to the column length. Values for NUMBER and FLOAT columns are checked for validity, and invalid numbers are set to NULL. Values for DATE columns are assumed to be in the internal time format and are checked for validity and set to NULL if invalid. Invalid column values are output to the log file.

The SQL Update statement is for use in tables with a set of key columns. If there is a row with the keys set to the values used in the statement, that row is updated. If there is no row with the same keys, the row is inserted. For example:

```
sqlupdate picklist_info_upd_tr |
       system_om, type_om, crew_restore |
       &System,   &Type,   @restoredTime |
       ref_id | $O.event.idx;
```

ensures that there is a row in the 'picklist_info_upd_tr' table with a 'ref_id' column equal to the outage's event index, and sets the 'system_om', 'type_om', and 'crew_restore' columns to the values in the 'System' element, 'Type' element, and 'restoredTime' variable, respectively.

The 'sqlupdate' introduces the SQL update statement, and is followed by the table or view name, followed by a vertical bar (|) as a separator. The list of non-key column names follows, separated by commas, and terminated by a vertical bar. The list of non-key data sources follow, separated by commas, and terminated by a vertical bar. The number of non-key sources must match the number of non-key columns. The list of key column names follows, separated by commas, and terminated by a vertical bar. The list of key data sources follow, separated by commas, and terminated by the statement terminator, ';'. The number of key sources must match the number of key columns.

The set of rules for Update statements is the same as described above for Insert statements above, except that nothing is written to the database if a key column is missing.

If Statement

If statements are used to alter the flow of document processing, allowing elements to be generated optionally in output XML, different tables to be updated depending on data in incoming XML, etc. For example:

```
if (@doctype != create)
{
   # there is no job number until the reply to the create document is
received
   &JobNumber = $O.externalId;
}
```

The statements between the brackets '{' and '}' are only processed when the variable 'doctype' does not contain the string "create".

An if statement can also have an 'else' clause. For example:

```
if (@v1 == @v2)
{
   logDebug(0, "v1 is " , @v1, " as is v2");
}
else
{
   logDebug(0, "v1 is ", @v1, " and v2 is ", @v2);
}
```

Note: An if statement can have any number of 'elseif' clauses, optionally followed by an 'else' clause. For example:

```
if (@a == 1)
{
   logDebug(0, "a is 1");
}
elseif (@a == 2)
{
   logDebug(0, "a is 2");
}
else
{
   logDebug(0, "a is ", @a, " neither 1 nor 2");
}
```

`'else if' (two words) can be substituted for 'elseif'.`

'if' statements can contain nested 'if' statements, for example:

```
if (@a == 1)
{
   if (@b == 1)
   {
   logDebug(0, "a is 1, b is 1");
   }
   else
   {
   logDebug(0, "a is 1, b is ", @b);
   }
}
elseif (@a == 2)
{
   if (@b == 1)
   {
   logDebug(0, "a is 2, b is 1");
   }
   else
   {
   logDebug(0, "a is 2, b is ", @b);
   }
}
else
{
logDebug(0, "a is ", @a, ", b is ", @b);
}
```

Flags

Flags are used to modify the behavior of elements, attributes and variables. Flags are set by adding a flag specification to the entity's definition. Individual flags have a flag character, used in flag specifications.

The flags are:

- The 'send on change' flag 'S'. This flag only applies to elements, attributes and triggers in Output Documents. It is described below in sections **Change Detection** on page 7-25 and **Triggers** on page 7-31.
- The 'always include' flag 'I'. This flag only applies to elements and attributes in Output Documents. It is described below in section **Change Detection** on page 7-25.
- The 'don't save' flag 'D'. This flag only applies to elements and attributes in Output Documents. It is described below in section **Change Detection** on page 7-25.
- The 'ignore attribute changes' flag 'A'. This flag only applies to elements in Output Documents. It is described below in sections **Change Detection** on page 7-25.
- The 'default to current' flag 'C'. This applies to elements, attributes and variables in Output Documents. Normally when the source of an assignment fails (due to the unavailability of an incident, for example) the element, attribute or variable is set to its default (described below). If this flag is on, the previous value is used instead. This only has effect when change detection (see below) is in effect.
- The 'required' flag 'R'. This flag applies only to elements and attributes in Input Documents. If this flag is set, the element or attribute must be present in the input XML for the Input Document to process the XML. If an attribute is required, the element itself is implicitly required.

Flags are set by adding a flag specification after the name in an element, attribute or variable declaration. For example:

```
&Elem:CSI = $E.devAlias;
```

The flag specification is the ':' followed by one or more flag characters.

All elements and attributes have definitions. However, variables do not have to be. They are implicitly defined when they are assigned in an assignment statement and in a SQL select statement. Such implicit definitions cannot have flag specifications. An example of a variable definition with a flag specification is:

```
@var:C;
```

Defaults

There are cases when the values for an element, attribute or variable are not available. In this case the values are set to the entity's default.

Values are not available in the following cases:

- When an optional element is not present in input XML.
- When insufficient incidents are associated with an event.
- When a table name is dynamically set, for example a device's facility table, and a column is referenced that the table does not contain.
- When a sqlselect statement selects no rows.

- A function call cannot generate a value, perhaps because it needs to use an external object that is not available. The specific situations are discussed in the function reference section of **DML Function Calls** on page 7-70.

If no default is specified, and the ‘default to current’ flag is not set, the default value for an entity is "", the empty string. The ‘default to current’ flag is discussed above in **Flags** on page 7-23.

A default is specified by adding a default specification to the entity’s definition, after its (possibly empty) flag specification. For example:

```
&CallerName::None = $I.0.getCustomerName;
```

This normally sets the ‘CallerName’ element to the name of the first customer that called. If the event has no incidents, the element’s value is set to ‘None’. In this case the flag specification ‘:’ is empty, and is followed by the default specification ‘:None’. If a change to the element ‘CallerName’ causes an update, the ‘S’ flag is included in the flag specification:

```
&CallerName:S:None = $I.0.getCustomerName;
```

This time the flag specification is ‘:S’ and the default specification remains ‘:None’.

Alternate Names

Alternate names can be given to elements, attributes, and variables. Their usage depends on the type of Document the entities are in.

Input Documents

In Input Documents, only elements can have alternate names. They are used to give the elements names, which are easier to use when the element is a deeply nested sub-element. For example:

```
&Grandparent
{
    &Parent
    {
        &son:::Jim<jattr1;>;
        &daughter;
    }
}
```

Declares an element ‘Grandparent’ with one sub-element ‘Parent’, which has, in turn, two sub-elements ‘son’ and ‘daughter’. ‘son’ has a name of ‘Jim’. The following example accesses the value of ‘daughter’:

```
@dval = &Grandparent/Parent/daughter;
```

while accessing the value of ‘son’ uses the shorter and less error prone:

```
@sval = &Jim;
```

To obtain the value of the attribute of ‘son’:

```
@sattr = &Jim:<jattr1>;
```

When an element has a name, the long form cannot be used.

Output Documents

In Output Documents elements, attributes, and variables can have alternate names. They have two usages:

- To allow other documents to set their values, for example using the function 'setDocValue', which needs the alternate name.

  An example of this is when an order contains the estimated time of restoration (ETR), which is initially supplied by the Oracle Utilities Network Management System services. Subsequently, the field crew, after assessing the situation in the field, decides that this time is inappropriate and sends a message back. This message causes input XML to be sent to the adapter, which will then be processed in an Input Document. The document contains an API call to set the ETR in Oracle Utilities Network Management System to this value.

  The ETR value in Oracle Utilities Network Management System now matches the ETR value in the MDS. However, the Output Document holds the value of all elements of the order sent to the MDS, to prevent unnecessary updates, and, in this example, will hold the ETR previously sent to the MDS.

  The consequence of this situation is that the next time the event associated with the order changes, the Output Document will be triggered and a change to the ETR will be detected, even if no other elements have changed. This will cause an unnecessary transmission of an order update to the MDS.

  The solution to this defeating of the change detection mechanism is to call 'setDocValue' with the new value, and the name of the element that contains the estimated time of restoration. Similar situations can arise for attributes and variables.

- To uniquely name elements so that their values can be saved to the database so that change detection can continue to be effective over a adapter shutdown/restart cycle. This is further discussed in section **MDS_ORDER_VALUE** on page 7-42.

  The name is specified in an alternate name specification following the (possibly empty) flag specification, and (possibly empty) default specification. For example:

  ```
  &ETR:S:"No ETR":etr = formatDateTime($E.estRestTime);
  ```

  This defines an element with the 'S' flag set, with a default of 'No ETR', and an alternate name of 'etr' which is set to the formatted value of the current event's estimated time of restoration.

Change Detection

When MDS orders are updated due to changes in the event that is associated with the order, it is important to send only the elements that have changed. This is because of the limited bandwidth available to transmit data to and from field crews.

In addition, some elements that change may not be important enough to cause an update to occur, for example, the number of customer calls that are associated with the event.

Some elements that do not change may have to be sent every time that an update message is to be sent, for example the order identifier could be the key on the MDS.

Change detection only applies to Output Documents, and only to documents that update data previously sent to the MDS. To indicate that change detection applies, change detection only occurs when the trigger invoking the Output Document has the 'S' (send on change) flag set. The rest of this section assumes that change detection applies.

The default setting for an element is that it is only included in the output XML when it changes, and that a change to the element is not important enough to cause an update to be in effect. For example:

```
&NumCustCalls = $E.custCall;
```

An element that causes the update to be sent has the 'S' (send on change) flag set. For example:

```
&IntDev:S = $E.devAlias;
```

An element that is always sent when the update is to be sent has the 'I' (always include) flag set. For example:

```
&Create:I < confirm="Always"; >;
```

The 'S' and 'I' flags can be combined. This means that the element is always sent on an update, and that a change to it causes an update to be sent. For example:

```
&JobNumber:SI=$O.externalId;
```

Change detection applies to element attributes in the same way as elements.

If the element or any attributes is to be included in the output XML, the element and all of its attributes (including those that have not changed) are included in the output XML.

When an object with an associated document is created, a copy is made of the document, with all elements and attributes marked as changed, because they have not been sent to the MDS yet. If an element is not included in the initial transmission of the document (due to an if statement), it, and all its attributes, remains marked as changed. Normally, this is the correct behavior. However, if it has constant attributes, and may is set externally by a call to setDocValue to suppress sending the value, the element will no longer be marked as changed, but the attributes will remain marked as changed. This means that the first time that the element is processed (due to the condition in the if statement changing), it will be transmitted because the attributes are marked as changed, which is likely not the intention.

Using the 'ignore attribute changes' flag 'A', defeats the behavior. It can be combined with the 'S' and 'I' flags, if appropriate. For example:

```
 if (($E.est_source == "O") || ($E.est_source == "C"))
{
   WorkCodeUDF:SA::ert = formatDateTime($E.estRestTime);
}
```

If the element or any attributes is to be included in the output XML, the element and all of its attributes (including those that have not changed) are included in the output XML.

The values of the elements and attributes are held in memory while the adapter is executing. In order to preserve these values over a shutdown/restart cycle, they are stored in the database. By default, the adapter saves all elements and attributes that do not have constant values. However, some element and attribute values are always set during the execution of the Output Document, for example:

```
if (@docType == "create")
{
   @conf = Always;
}
else
{
   @conf = Never;
}
&Confirm:I = @conf;
```

In this example, there is no need to save the value of the 'Confirm' element. To prevent saving its value, add the 'don't save' flag 'D' to the element's definition, for example:

```
&Confirm:ID = @conf;
```

The table used to save the values of the order document is described in section **The Order Tables** on page 7-41.

The External Objects

The external objects are:

- The Event object, identified by 'E', holds the data from the event's latest 'SRSoutput' data. The names of the fields are the relevant fields in the Oracle Utilities Network Management System SRSoutput class. They are listed in **DML Function Calls** on page 7-70.
- The Incident object, identified by 'I', holds the data from the event's customer calls. The Incident object is not automatically populated. It can be populated by a call to the function `readIncidents()`. If memory is at a premium, the Incident object can be unpopulated by the function `clearIncidents()`. The names of the fields are the relevant fields in the Oracle Utilities Network Management System Incident class. They are listed in **DML Function Calls** on page 7-70. There can be any number of incidents associated to an event, including zero. Therefore, the offset of the incident of interest has to be supplied when accessing an incident field. Offsets start at zero. The offset is an integer, and is the second component of the external data reference. For example, `$I.1.getCustomerName` reads the name of the second customer in the incident array. A function, `sortIncidents()`, is supplied to sort the incidents. Before sorting they are in the order that the customers called. Any reference to an incident field can fail, because there may be no incidents. Such a failure causes the value of the element, attribute or variable being assigned to be set to its default.
- The Order object, identified by 'O', holds data relating to orders sent to the MDS. It holds the order's current event object (possibly aggregated to summarize a set of related events), the Output Document that holds all the document data for the order for change detection, a number of fields that are always available (section **Permanent Order Object Fields** on page 7-135, **Permanent Order Object Fields** on page 7-135 of **DML Function Calls** on page 7-70), and other data fields as configured by the MDS_ORDER and MDS_ORDER_FIELD tables. See **MDS_ORDER** on page 7-41 for a description of the order tables.
- The Relationship object, identified by 'R', holds data relating to event relationships processed by the dml. It holds the type of relationship, the handles of all the related events, and a number of fields that are always available (see section **Permanent Relationship Object Fields** on page 7-135, **Permanent Relationship Object Fields** on page 7-135 of **DML Function Calls** on page 7-70).
- The Global Data object, identified by 'G', which holds named values for use by any of the documents. A field is created by assigning values to the field, e.g., `$G.QueueManager = OPS;`. If a field that has not been created is read, the field is created with a value of "", the empty string, and an warning is output to the log. The global data object usually holds configuration data, and the fields are set in a Configuration Document.
- The Trigger Parameter object, identified by 'T', which holds the parameters sent to Output Documents when they are triggered from another document using the function 'triggerOutputDoc'. The first parameter to 'triggerOutputDoc' specifies the Output Document to trigger, the second parameter specifies the trigger to pull, and the rest of the parameters values are available to the Output Document as fields '1', '2' … up to the number of parameters supplied, less two (the document and trigger names). For example, `@param1 = $T.1;` sets the variable 'param1' to the value of the first parameter.

Input Element Arrays

The declaration of input elements discussed in section **Input Elements and Attributes** on page 7-19. can only handle elements whose tags are unique within the bounding parent element (which may be the root element). If such an element appears in the input XML, its value (or those of its own sub-elements) will override the values previously read. Input element arrays solve this problem.

Generally there are two situations that repeated elements are required:

- When the elements contain the same type of data and each should be processed in a similar fashion. For example, a crew logs on when the MDS had previously assigned multiple orders to the crew. Each of these assignments should be made to the crew on Oracle Utilities Network Management System.
- When the elements contain different types of data, and the individual elements are identified by the value of a particular attribute, in effect giving the element a compound tag.

Using an element array with an unspecified index solves the first situation. For example:

```
&CrewAssignments
{
&CrewAssignment[]
{
&OrderId;
&AssignmentStatus;
}
}
```

This allows any number of elements with the tag 'CrewAssignment' to be processed. This can be achieved using the 'for' statement described below.

Using an element array with a specified index attribute solves the second situation. For example:

```
&UDFS
{
&UDF[idx];
}

@etr = &UDFS/UDF[et];
@troubleType = &UDFS/UDF[tt];
```

which assigns the 'UDF' element whose 'idx' attribute is 'et' to the variable 'etr' and the 'UDF' element whose 'idx' attribute is 'tt' to the variable 'troubleType'.

Array elements can be given alternate names in the same way as normal input elements, but they cannot be given flags and defaults, as they have no meaning. Individual array indices can be required to be present, using a required index specification. For example:

```
&UDFS
{
&UDF[idx] R(et, tt);
}

@etr = &UDFS/UDF[et];
@troubleType = &UDFS/UDF[tt];
```

Prevents the processing of the input XML when the input XML does not contain a 'UDF' element with an 'idx' attribute of 'et' and contain a 'UDF' element with an 'idx' attribute of 'tt'. The required index specification consists of an 'R' followed by a comma separated list of attribute values in parentheses '(' ')'.

While sub-elements to array elements can be given names, their usage differs from normal sub-elements because the appropriate index must be specified. This is done using a format similar to the long form of referencing tags, using forward slashes '/'. For example:

```
&ancestor
{
    &grandparent[idx]
    {
        &parent
        {
        &child;
        }
    }
}
@son = &ancestor/grandparent[smith]/parent/child;

uses the full long form.

&ancestor
{
    &grandparent[idx]:::granny
    {
        &parent
        {
        &child;
        }
    }
}
@son = &granny[smith]/parent/child;

uses a name for the array element.

&ancestor
{
    &grandparent[idx]:::granny
    {
        &parent
        {
        &child:::son;
        }
    }
}
@son = granny[smith]/son;
```

uses a name for both the array element and its sub-element.

For Statement

The for statement, or for loop, is used to iterate through the contents of an array element.

The array element can either have a specified index attribute or an unspecified index. The usage is similar.

For example:

```
&CrewId:R;

&CrewAssignments
{
&CrewAssignment[]:::asn
{
&OrderId;
```

```
&AssignmentStatus;
}
}
if (!findCrewById(&CrewId))
{
    stop;
}

for (&asn[], @i)
{
    if (isSet(&asn[@i]/OrderId) &&
     isSet(&asn[@i]/AssignmentStatus) &&
     findOrder(externalId, (&asn[@i]/OrderId))
    {
        if (&asn[@i]/AssignmentStatus == ASN)
{
assignCrew();
        }
elseif (&asn[@i]/AssignmentStatus == DSP)
{
dispatchCrew();

}
    }
}
```

The for statement executes all the statements between the matching brackets '{' and '}', for each of the elements in the tag array ('&asn[]' in this case), in the order that they appeared in the input XML. During execution of the statements the 'for' variable ('@i' in this case) is set to the elements index in the tag array. As this array did not have an index attribute specified, the variable has a numeric value, starting at zero, and is incremented by one between loop executions. If the XML does not contain the element, the loop is never executed.

If, instead, the array element had an index attribute, for example:

```
&CrewAssignments
{
&CrewAssignment[idx]:::asn
{
&OrderId;
&AssignmentStatus;
}
}
```

the loop variable would contain the values for the 'idx' attribute included in the input XML. Note, however, that if the XML contained any 'CrewAssignment' elements, without an 'idx' attribute, these elements would be ignored, and therefore not available for processing.

Queue Specification

The queue specification is used to specify the queue that XML from Output Documents are sent to, and the queue that input XML is received on to be processed by Input Documents. The queue specification appears in the header portion of the document. For example:

```
queue = NMS.TO.MDS.REQUEST;
```

Which sets the queue name to a constant. Commonly, the queue is set to the value of a field in the global object to facilitate the configuration of the queue names.

Output Documents can change the queue name by accessing the queue variable in the body of the document. For example:

```
If (@use_alternate_queue)
{
   @queue = $G.alternateQueue;
}
```

Association of an Output Document to the Order Object

The order document must have an associated Output Document that generates the Order creation (usually) and update (always) XML. It is used to cache all the element data that is used for change detection. This association is achieved using the associate specification in the document header which takes the form:

```
associate = O;
```

where the ‘associate’ is the associate keyword and the ‘O’ identifies the Order Object.

Triggers

Triggers specify when Output Documents are activated to generate XML to be sent to the MDS. For example:

```
trigCreateOrder < SRSoutput; > =
($E.status == "ACK") && !findOrder(event, $E.outageHdl);
```

might be used to trigger the creation of an order.

The ‘trigCreateOrder’ is the trigger’s name. The value of a trigger is available to the body of the document in a variable with this name.

The angle brackets, ‘<’ and ‘>’, contain the trigger specification(s), which define the circumstances under which the trigger is to be tested. In this case the arrival of an asynchronous ‘SRSoutput’ message.

After the assignment is the expression which determines when the trigger is to be fired. The expression is evaluated and if it is true, i.e. not the empty string, the trigger is fired. The result of this evaluation is then assigned to the trigger variable. In the example, the trigger will fire if the event is in the acknowledged state and no order has been created for the event.

If an expression is not supplied, the trigger always fires when the trigger specification is satisfied. In this case, its value is set to ‘Y’.

Triggers can also have the ‘send on change’ ‘S’ flag set. If not set, all elements in the document are sent in the resulting XML. If set, only those elements that have changed are set. For example:

```
trigUpdateOrder:S < SRSoutput; > =
(!isIn($E.status, "UNA", "CNL") && findOrder(event, $E.outageHdl);
```

There can be multiple triggers, only one of which is fired. The triggers are evaluated in order of their definitions and the first to fire takes effect. **The unfired triggers all have a value of false.**

There are a number of types of trigger specifications. In some cases they need values. They are:

- The reception of a SRSoutput message. The trigger specification is ‘SRSoutput’, which has no value. Examples of the SRSoutput trigger are shown above.

- When an event is deleted. This occurs when outages are merged, and when a previously processed event is not returned by SRSService at start up. The trigger specification is '**EventNonexistent**', which has no value. Commonly, this triggers the same Output Document that handles event cancellation. The event is supplied in trigger parameter 1. An example of an event non-existent trigger is:

```
triggerEventNonexistent<EventNonexistent;>
                =  findOrder(order, $T.1) && $O.externalId;
```

- Periodically. The trigger specification is 'Periodic', which needs a value that is the period at which to fire the trigger, in seconds. An example of a periodic trigger that fires once a minute is:

```
trig1min< Periodic=60; >;
```

- By request in another document, using the 'triggerOutputDoc' function. The trigger specification is 'OnRequest', which needs a value that is the number of trigger parameters that need to be passed by the requesting document. An example of a request trigger with two parameters is:

```
trigOnRequest<OnRequest=2;>
                =  findOrder(event, $T.1) &&
                   isIn($E.status, "UNA", "CNL") &&
                   $O.externalId;
```

- Creation of, changes to the number of events in, and deletion of an event relationship processed by dml. The trigger specification is 'RelationChanged', which as no value. The Output Document is activated once for each event in the relationship. The event is supplied in trigger parameter 1. An example of an relationship change trigger is:

```
trigRelationChanged< RelationChanged; >
= findOrder(event, $T.1) &&
          isIn($E.status, "UNA", "CNL");
```

- Deletion of an event relation processed by dml. The trigger specification is 'RelationDeleted', which as no value. The Output Document is activated once for the relationship. The relation is supplied in trigger parameter 1. An example of an relationship deletion trigger is:

```
trigRelationDeleted< RelationDeleted; >
= findOrder(RELATED_OUTAGE, relation, $T.1) &&
        $R.externalId;
```

- A change to the number of events in an event relation that is aggregated by the adapter. The trigger specification is '**AggregateChanged**', which has no value. The Output Document is activated once for the order in which the events are aggregated. The order is supplied in trigger parameter 1. An example of an aggregate change trigger is:

```
trigAggregateChanged< AggregateChanged; >
= findOrder(order, $T.1) &&
      $O.externalId;
```

  This trigger is often used to ensure that all events in the relation have data updates sent by the MDS before new events are added to the relationship, for example the estimated time to restore.

- An SRSoutput message arrives indicating **number of events are merged**. The trigger specification is '**EventMerged**'. Its value specifies the name of an Output Document, known as the Merge Priority Document, which is used to generate the priority of all of the orders associated with the events involved in the merge, if any. The Merge Priority Document must have a request trigger named ‘call’. The Output Document that contains the event merged trigger is known as the Merge Document. There can be zero or one Merge Document in all the dml files processed by the adapter. If no Merge Document is supplied, the SRSoutput message is processed via SRSoutput triggers, and the merged events are processed via event non-existent triggers. If a Merge Document is supplied, the adapter processes the SRSoutput message as follows:

1. For each of the events involved in the merge, including the surviving event, the adapter determines whether the event is associated with an order. If so it’s order becomes eligible to be the surviving order, except for a non-surviving event’s order is in an aggregate relationship. If there are no eligible orders, the SRSoutput message is processed via SRSoutput triggers, and the merged events are processed via event non-existent triggers. Neither the Merge Document nor the Merge Priority Document is processed.
2. When there are multiple eligible orders, the adapter must choose which one will become the survivor. The Merge Priority Document provides a priority value to allow this choice to be made. The adapter processes the Merge Priority Document via the ‘call’ trigger. The Merge Priority Document has its order and event objects set. Note that if the event is the surviving event, the previous version of the event object is set. This allows the event object’s fields before the merge to be used to determine the priority.
3. The Merge Priority Document returns the priority in the trigger argument object. The priority consists of a priority sort definition and one or more priority values. The priority sort definition is returned in the ‘0’ (the character zero) field of the trigger argument object ($T.0). If this value is the empty string, the order is no longer eligible to survive. If not empty, it defines the number and sort order of the priority values. The priority values are returned the ‘1’, ‘2’, … fields ($T.1, $T.2, …), up to the number of values. The number of characters in the priority sort definition determines the number of values expected by the adapter. Each character determines the sort order for the corresponding value, ‘A’ or ‘a’ for ascending and ‘B’ or ‘b’ for descending, the first character applies to $T.1, the second to $T.2, etc.
4. If there are multiple orders eligible, the one with the highest priority becomes the survivor. Each priority value is sorted alphabetically in the order specified by its character in the priority sort definition. The first value ($T.1) is examined first. If one order ranks higher that all the other, it is chosen. Otherwise, any order with a lower priority is discarded. Then the other values are examined in turn, until an order is chosen. If all the priorities are the same, the order associated with the surviving event, if any, survives. If the surviving event has no order, the order associated with the oldest event survives. For example, the following dml fragment illustrates how to choose the order with the most advanced crew assignment/dispatch status, and if they are equal the order with the oldest event.

```
# two priority values, the first descending, the second ascending
$T.0 = DA;

# orderCrewStatus() returns ’A’ for Assigned, ‘D’ for dispatched,
  ‘O’ for on site, nothing for none
@ocr = orderCrewStatus();
if (@ocr)
{
$T.1 = @ocr;
}
else
{
# ‘a’ precedes all of the valid values
$T.1 = a;
}
```

```
# formatDateTime() returns the date and time in the format
# YYYY-MM-DDTHH:MM:SS
$T.2 = formatDateTime($E.outageTime);
```

5. The Merge Priority Document can populate other fields in the trigger argument object, to be used by the Merge Document when the order being processed becomes the surviving order. Note that the Merge Priority Document is processed even if there is only one eligible order so that these fields, if any, can be passed to the Merge Document.
6. If the surviving order is not associated with the surviving event, the surviving order is re-associated to the surviving event. The order that was associated with the surviving event is re-associated to the surviving order's old event, so that it can be processed via an event non-existent trigger, to allow normal order clean up.
7. The adapter processes the Merge Document with the surviving order object and surviving event object set to the new version of the event, and the trigger argument object fields set by the Merge Priority Document when processing the surviving order.
8. Finally, the SRSoutput message is processed via SRSoutput triggers, and the merged events are processed via event non-existent triggers.

An example of an event merged trigger is:

```
TrigEventMerged <EventMerged="OrderMergePriority";>;
```

For this to be valid, there must be an Output Document named 'OrderMergePriority', with a request trigger named 'call'.

The Root Element

The root element is the element that contains all of the other elements in an XML document. The root element in dml serves a different purpose depending on whether it is part of an Output or Input Document.

In an Output Document, it is used to generate the root element in the XML to be sent. In an Input Document it is used to select the Input Document or Documents that the input XML can be processed by, similar to a trigger for an Output Document. An example of a root element follows:

```
&RootElement <environment = $G.environment; revision = "1.0.0";> =
CreateJob;
```

The root element has the same format for both Input and Output Documents.

The root element differs from other elements in that the element's tag is defined by the value of the element, 'CreateJob' in this example. The pseudo-tag 'RootElement' keyword identifies it to be the root element.

In an Output Document, the example would generate the following start element tag:

```
<CreateJob environment="Test" revision="1.0.0">
```

assuming that the global field 'environment' held the value 'Test'.

To select an Input Document, the input XML's root element must have the same tag, and all the attributes in the root element specification must be present and have exactly the same contents. Extra attributes in the XML's root element are ignored.

An Input Document with the same root element definition example above would be selected by the root element in the example above.

The root element is defined in different areas, depending on the type of document.

In an Output Document the root element is in the body of the document, because it's tag may have to be determined during document processing. It is the only element in an Output Document that is an exception to the rule that states that elements are generated in the order that they appear in the Output Document.

In an Input Document the root element is the header of the document because it is used before any processing is done in the document.

The Base Path

The base path only applies to Input Documents, and is not necessary, but can make Input Documents more compact. Consider the following XML:

```
<ConfirmJob>
 <ApplicationArea>
 </ApplicationArea>
 <DataArea>
  <Job>
   <OriginalApplicationArea>
    <BODId>OrigBODId</BODId>
   </OriginalApplicationArea>
   <CreateSuccess>
    <JobNumber>1234567890</JobNumber>
   </CreateSuccess>
  </Job>
 </DataArea>
</ConfirmJob>
```

The XML could contain many other elements, but the 'BODId' and 'JobNumber' are the only elements to be processed. The elements in the Input Document would be:

```
&DataArea
{
    &Job
    {
        &OriginalApplicationArea
        {
        &BODId;
        }
        &CreateSuccess
        {
        &JobNumber;
        }
    }
}
```

Note that 'BODId' and 'JobNumber' have a common grandparent 'Job'. By setting the base element to the grandparent by adding:

```
BasePath = DataArea/Job;

to the document's header, the elements would become:

&OriginalApplicationArea
{
   &BODId;
}
&CreateSuccess
{
   &JobNumber;
}
```

Stop Statement

The Stop statement causes the processing of the current document to stop. It takes the form:

```
stop;
```

It is usually contained in an 'if' statement. Any statements with side effects, for example a SQL Insert, processed before the 'stop' do take effect.

A stop statement prevents an Output Document's XML from being sent.

A stop statement in a Configuration Document, described below, causes initialization to fail.

Include Statement

Sometimes it's useful to have the same set of statements in two different places in a dml file. For example all Output Documents may need the same header, or application area. This can be achieved by placing the repeated statements in another file and then including the file more than once in another file. The following is an 'include' statement:

```
include ohdr.dml
```

This 'include' statement in effect replaces the text 'include ohdr.dml' with the contents of the file 'ohdr.dml'. Files being included can also include other files, but the nesting level is limited to 10 deep so that infinite recursion can be prevented.

Configuration Documents

Configuration Documents are used to set configuration data and load configuration tables from the database at initialization time. They are similar to Input Documents, but have no elements.

They are processed at initialization time and then discarded. Any errors encountered when processing a Configuration Document should be made fatal by executing a stop statement, causing initialization to fail.

Common uses are to set fields in the global data object, including configuration fields, and to load mapping tables, used by the ‘mapTableStr’ and ‘mapTableCode’ functions described in **DML Function Calls** on page 7-70. For example:

```
ConfigDoc Configure
{
# fields for other documents
$G.OutRequestQueue=NMS.MDS.REQUEST;
$G.OutErrorQueue=NMS.ERROR;
$G.InReplyQueue=MDS.NMS.REPLY;
$G.InRequestQueue=MDS.NMS.REQUEST;
$G.environment=Test;

# configuration parameters
$G.config_QueueManager_name=OPS;
$G.config_OutQueue_req_name=$G.OutRequestQueue;
$G.config_OutQueue_err_name=$G.OutErrorQueue;
$G.config_OutQueue_numThread=5;
$G.config_InQueue_rep_name=$G_InReplyQueue;
$G.config_InQueue_rep_numThread=2;
$G.config_InQueue_req_name=$G.InRequestQueue;
$G.config_InQueue_req_numThread=4;
$G.config_ErrorQueue_name=$G.OutErrorQueue;
$G.config_ErrorDoc_name=Error;
$G.config_ErrorDoc_trigger=xmlErrorTrigger;

$G.config_Relation_Aggregate_AcknowledgeEvents = ack;
$G.config_Relation_Aggregate_type = PARTIAL_RESTORATION;
$G.config_Relation_Aggregate_ActiveEvents = Y;
$G.config_Relation_dml_type = RELATED_OUTAGE;


# Load map tables
if (!loadMapConfigTable(mds_map_config) ||
    !loadMapTable(mds_cls_desc) ||
    !loadMapTable(mds_cls_type))
{
   stop;
}
}
```

Configuration Field Reference

Configuration fields have names starting with ‘config_’. The available configuration fields are:

- **config_QueueManager_name**: The name of the queue manager to use. This field must be specified, or the adapter will exit with a configuration error message.
- **config_OutQueue_<id>_name**: The name of an output queue to use. Each output queue needs to have a unique id, which replaces the ‘<id>’. The unique id may not contain an underscore (_).
- **config_OutQueue_numThread**: The number of threads to use to generate output documents. If not specified, one thread is used to generate output documents.
- **config_InQueue_<id>_name**: The name of an input queue to use. Each input queue needs to have a unique id, which replaces the ‘<id>’. The unique id may not contain an underscore (_).
- **config_InQueue_<id>_numThread**: The number of threads to use to process input documents arriving on the input queue with the same id. Each input queue needs to have a unique id, which replaces the ‘<id>’. If not specified, one thread is used to process documents on the queue.
- **config_ErrorQueue_name**: The name of the queue to send XML parse error and warning reports to. This field must be specified, or the adapter will exit with a configuration error message.
- **config_ErrorDoc_name**: Specify an output document to process error reports. If specified, the config_ErrorDoc_trigger field must be specified. If not specified, the standard XML error document, as specified in the MQ/XML adapter documentation, is used.
- **config_ErrorDoc_trigger**: The trigger to pull in the error document when there is an error or warning to report. Ignored is no error document is specified. If the specified trigger does not exist in the error document, the adapter exits with a configuration error message. The adapter supplies three trigger arguments to the error document. Argument one is the error or warning message description, argument two is the priority level, one of ‘Warning’, ‘Error’, and ‘Fatal Error’, and argument three is the offending XML document.
- **config_Relation_Aggregate_type**: A comma separated list of relationship types to aggregate. The relationship types are: NESTED_OUTAGE, MOMENTARY_OUTAGE, PARTIAL_RESTORATION, and RELATED_OUTAGE. If a pseudo relationship of a type that is aggregated by the adapter is created using createPseudoRelation(), the pseudo relation is aggregated. If a relationship type is configured for both aggregation and dml processing, the adapter exits with a configuration error.
- **config_Relation_Aggregate_AcknowledgeEvents**: Acknowledge all unacknowledged events in aggregate relations. The value of this field is used in a call to SRS::requestRowAction() as the button name. This field should be set if pseudo aggregate relations are created on events that may not be acknowledged.
- **config_Relation_Aggregate_ActiveEvents**: If the field is not the empty string "", sum count data for all active events in an aggregated relation. The count data event fields are: customersOut, crit_k, crit_c, and crit_d.
- **config_Relation_dml_type:** A comma separated list of relationship types to be processed by the dml, using the relationship object. The relationship types are: NESTED_OUTAGE, MOMENTARY_OUTAGE, PARTIAL_RESTORATION, and RELATED_OUTAGE. If a pseudo relationship of a type that is processed by the dml is created using createPseudoRelation(), the pseudo relation is processed by the dml. If a relationship type is configured for both aggregation and dml processing, the adapter exits with a configuration error.

- **config_Relation_dml_AcknowledgeEvents**: Acknowledge all unacknowledged events in dml processed relations. The value of this field is used in a call to SRS::requestRowAction() as the button name.
- **config_MaxBackoutCount**: The adapter uses the MQSeries syncpoint facilities to preserve input messages when there is a failure. Using the MQSeries syncpoint facilities introduces the possibility that a 'poison' message will be sent by the MDS. A 'poison' message is one that can never be successfully processed, for example because an Oracle Utilities Network Management System table has a constraint that the contents of the message violate. If the 'poison' message is never discarded, MDS will continually try to process the message, fail, and then restart. This parameter sets a limit on the number of restarts that MDS will perform before discarding a message. The default value is five. A value of zero disables this feature. If a message is discarded, the error is logged to the log file, including the offending XML, and if the error document is configured, and an error report is sent to the error queue.
- **config_Event_QueueDelay**: When event data is received in an SRSoutput message, from SRSService, they are held for this period before they are queued for processing. If another message for the same event is received before the delay has expired, the older message is discarded, and the new message is held for the delay period. This avoids unnecessary processing and message transmission when event data is changing rapidly. Setting this delay too short can cause extra messages, while setting it too long can cause poor response. The value is in seconds. The default value is four seconds. A value of zero disables this feature.
- **config_Event_ReprocessPeriod**: In certain circumstances, for example when more that one event is grouped, the adapter needs to request an event's status from SRSService. The adapter periodically requests these events' statuses in a single request, reducing the burden on SRSService. In addition, if an SRSoutput message for one of these events arrives before it is time to do the request, the request does not have to be made. Setting this period too short can cause extra messages, while setting it too long can cause poor response. The value is in seconds. The default value is six seconds. The minimum value is two seconds. It must be at least 2 seconds longer than config_Event_QueueDelay.
- **config_Crew_AssignmentCheckPeriod**: When the assignment of crews to events is the responsibility of the adapter, as a proxy for the MDS, Oracle Utilities Network Management System can be configured to reduce the possibility of an operator inadvertently assigning a crew to an event. However, it must be possible to assign crews if Oracle Utilities Network Management System cannot communicate with the MDS, hence mistakes can happen. The adapter periodically checks all crew assignments. A sub-set of crew assignments is checked at the end of each period, the events in ten orders, and ten crews for events not in an order being checked each time. Setting the period too short causes unneeded processing, setting it too long delays such mistakes being repaired. The value is in seconds. The default value is six seconds. A value of zero disables this feature.
- **config_Crew_MoveAssignmentCheckDelay**: When crew assignments are being checked, allowing the checking to occur during event grouping can, in some configurations, cause unnecessary message traffic between the adapter and SRSService. To prevent this, when a grouping happens, crew assignment checking is delayed. The value is in seconds. The default value is two seconds. A value of zero disables this feature.
- **config_Relation_CheckPeriod:** When event relationships are processed by the adapter, the adapter loads the relationship database table each time a relationship is created, changed or deleted. In addition, the adapter checks for changes to the table at this period. The value is in seconds. The default value is 30 seconds. A value of zero disables this feature.
- **config_IgnoreCondStatus**: In most implementations of the adapter, additional alarms in the Oracle Utilities Network Management System Work Agenda should not be processed. The value of this configuration parameter is a list of condition statuses, separated by commas (,), that are to be ignored by the adapter. To ignore additional alarms (often NFY events) set this value to 12.

- **config_CompleteStatus**: In certain situations the adapter need to be able to determine if an event is complete, for example to prevent the assignment of a crew to a completed event, which is illegal. This can only be determined by examining the status string for the event, which is configurable in Oracle Utilities Network Management System. The value of this configuration parameter is a list of statuses, separated by commas (,), that indicate that an event is complete. The default value of this parameter is CMP, CNL.
- **config_EventUpdateTimeout**: When device outages are confirmed and restored (using confirmDeviceOutage() and restoreOutage()), the new state of the event(s) needs to be read from SRSService. Because device operations are initiated by a message to DDService, which sends a notification to MTService to update the model, which subsequently notifies SRSService of the model change, the new state cannot be read immediately, because SRSService may not have received the notification. To avoid this, the adapter waits for an SRSoutput message updating the event before reading the new state. To prevent the adapter from hanging if an event update does not occur, a timeout is used to interrupt the wait. This value is the timeout in seconds. The default value is 20 seconds. The minimum is two seconds.
- **config_MaxThreadBusyUntilFatalError**: The adapter monitors the input and output threads to detect Mutex deadlocks, which would cause the adapter to hang. This value is the maximum number of seconds that a thread can be responding to a single trigger or input XML document. The default value is 300 seconds (5 minutes). The minimum is 60 seconds.
- **config_StopServiceOnHighLevelStop**: The adapter runs as a Windows Service, and it usually set to restart after a period after it fails. (This period is often the minimum one minute). If the adapter tells Windows that it has stopped normally, using a Service Stop message, just before exiting Windows does not try to restart the adapter. If the adapter does not send a Service Stop message, Windows will restart the adapter. When the adapter exits due to Stop request from the Service Property dialog, it sends Windows a Service Stop message, because the user is intending that the adapter stops and does not restart. When the adapter exits abnormally, it does not send a Service Stop message so that Windows will restart the adapter. However, when the adapter is sent a high-level stop message from Oracle Utilities Network Management System using the Action command, the user may or may not want the adapter to restart. This value determines whether the adapter sends a Service Stop message under these circumstances. A value of 'Y' causes the adapter to send a Service Stop message, while a value of 'N' prevents the adapter from sending a Service Stop message. The default value is 'N'.
- **config_AllowCloseOutEventCancel**: When this parameter is set to 'Y,' the DML function closeOutEvent will cancel the event instead of completing it, if the appliedRule value OUTAGE_PND_COMPLETE (26) is passed to the function. When this parameter is set to 'N,' the DML function closeOutEvent will not cancel events. The default value is 'Y'.
- **config_IgnoreStormmanUpdates**: When this parameter is set to 'Y' (default), the adapter will not process TRBL_ERT_UPDATE messages generated when **Storm Management** recalculates ERTs. When this parameter is to set to 'N' the adapter will process such messages.
- **config_AllowManualEntryForSCADA**: When this parameter is to set to 'Y,' the adapter will perform manual entry in order to operate a SCADA device in NMS model. When this parameter is to set to 'N' (default), the adapter will not be able to operate SCADA devices.

Pseudo Relationships

Pseudo relationships can be created by the dml so that all the events on a single device can be processed in the same way as an Oracle Utilities Network Management System created event relationship. The processing of these relationships is configured in a configuration document.

For example, a probable device outage is created from multiple customer calls grouping to the common transformer, but the crew discovers that there are multiple service problems only affecting some of the customers (perhaps a tree limb took down two service wires). The crew will fix all of the problems, so they do not need multiple MDS orders. When the individual service status is set for the affected customers, all the events generated will be treated in the same way as a partial restoration. (This example assumes that partial restorations are aggregated, and the dml creates a pseudo partial relationship when this situation arises.)

When all of the events in a pseudo relation are completed, the relation itself is automatically completed.

Configuration Tables

The following tables are used in conjunction with the dml files to configure the MDS adapter. They are loaded at startup, or, in some cases, when a dml statement needs them.

The Order Tables

The order tables save order data and configure the fields that are available to the dml in the external Order object, identified by ‘O’ in the dml.

MDS_ORDER

The MDS_ORDER table is used to save order data so that the adapter can continue to process orders over a adapter shutdown restart cycle. While not strictly a configuration table, it is described here to clarify the use of other configuration tables. The table has a fixed set of columns that are always in the table, and columns to hold Order object field values, element values and attribute values. The fixed set of columns is:

Column	Type	Description
h_cls	NUMBER	Order class
h_idx	NUMBER	Order Index
event_cls	NUMBER	Key event class
event_idx	NUMBER	Key event index
active_event_cls	NUMBER	Event class for last active event
active_event_idx	NUMBER	Event index for last active event
active	CHAR(1)	Active flag (Y = active, N = inactive)
when_xml_saved	DATE	When the xml data last sent to the MDS was saved in the database.
when_created	DATE	When the order was created
when_completed	DATE	When the order was completed or cancelled
comp_reason	VARCHAR2(64)	Text explaining why the order was completed or cancelled

The ‘h_cls’ and ‘h_idx’ columns make up the order handle, which identifies the order internally and are the key columns of the table. The order handle is available to the dml with a field name of ‘order’. The ‘event_cls’ and ‘event_idx’ columns make up the handle of the key event for the order. If the event is not aggregated, this is the single event associated with the order. If the event is aggregated, this is the key event of the relationship that makes up the aggregate. The event handle is available to the dml with a field name of ‘event’. The ‘active’ flag indicates whether the order is active, i.e. it has been created, but has neither been completed or cancelled. ‘when_created’ holds the time and date the order was created. ‘when_completed’ holds the time and date the order was completed or cancelled. comp_reason hold the text explaining why the order was completed or cancelled. This text is supplied by the dml when it completes the order by calling the ‘orderComplete’ function.

The columns that hold the field, element and attribute values can have any name, but they must match the contents of the MDS_ORDER_FIELD table and MDS_ORDER_VALUE table. Both are described below. It is suggested that the columns be named similarly to the name of the entity, allowing for the requirements of column names (case insensitivity, etc.). The column types should be VARCHAR2. The columns should be wide enough to hold the data that will be stored in them.

The off-line program that allows dml files to be checked before use can generate suggested contents for the MDS_ORDER, MDS_ORDER_FIELD, and MDS_ORDER_VALUE tables.

MDS_ORDER_FIELD

The MDS_ORDER_FIELD table maps the names of the fields in the Order object to the columns used to save their values in the MDS_ORDER table. In addition, it defines the names of all the fields that are available in the Order object. If the dml references a field that is not in this table, the adapter will log an error and initialization fails. The columns in the table are:

Column	Type	Description
name	VARCHAR2(32)	The field’s name.
col	VARCHAR2(32)	The column in the MDS_ORDER table that holds the field’s value.

MDS_ORDER_VALUE

The MDS_ORDER_VALUE table maps the names of the elements and attributes in the order XML to the columns used to save their values in the MDS_ORDER table. All elements and attributes except those with constant values, and those with the ‘don’t save’ flag set, must be included in this map. The use of the ‘don’t save’ flag is described in section **Change Detection** on page 7-25. The columns in the table are:

Column	Type	Description
name	VARCHAR2(32)	The element’s or attribute’s name.
col	VARCHAR2(32)	The column in the MDS_ORDER table that holds the element’s or attribute’s value.

The recommended way of naming the elements and attributes is to use the alternate name described in **Alternate Names** on page 7-24. If an alternate name is not specified, the adapter generates element names by numbering all the elements in the order that they appear in the Output Document and appending this number to the letter ‘e’, for example e1, e2. Note that all elements to be saved from the document are counted, including those with an alternate name, so that if a name is given to an unnamed element, all the other unnamed elements have the same name. If an attribute has no alternate name, the adapter generates the name by appending the underscore character '_' and the attribute's name to the attribute's element name, for example 'el_attr' is the name of the 'attr' attribute of an element with the alternate name 'el'.

An Example

To illustrate the configuration of the order tables and example tables are shown below, based on the contents of the Output Document in **DML Examples** on page 7-50.

The MDS_ORDER table schema:

Column	Type
h_cls	NUMBER
h_idx	NUMBER
event_cls	NUMBER
event_idx	NUMBER
active	CHAR(1)
when_xml_saved	DATE
when_created	DATE
when_completed	DATE
comp_reason	VARCHAR2(64)
BODID	VARCHAR2(128)
E1	VARCHAR2(64)
E2	VARCHAR2(64)
E3	VARCHAR2(64)
E4	VARCHAR2(64)
E5	VARCHAR2(64)
E6	VARCHAR2(64)
E7	VARCHAR2(64)
E8	VARCHAR2(64)
E9	VARCHAR2(100)
E10	VARCHAR2(64)
E11	VARCHAR2(64)
E12	VARCHAR2(64)
E13	VARCHAR2(64)
E14	VARCHAR2(64)
E15	VARCHAR2(64)
E16	VARCHAR2(100)

Column	Type
E17	VARCHAR2(64)
E18	VARCHAR2(32)
E19	VARCHAR2(32)

The contents of the MDS_ORDER_FIELD table:

col	name
BODID	BODId

The contents of the MDS_ORDER_VALUE table (the element tag is not in the table, it is included here to illustrate the numbering of the elements):

col	name	Element tag
E1	e1	Component
E2	e2	Confirmation
E3	e3	AuthorizationId
E4	e4	CreationDateTime
E5	e5	BODId
E6	e6	ExternalNumber
E7	e7	CreationDateTime
E8	e8	Device
E9	e9	Address
E10	e10	DevPhases
E11	e11	FuseSize
E12	e12	WinterLoad
E13	e13	SummerLoad
E14	e14	NumCustOut
E15	e15	CallerName
E16	e16	CallerAddr
E17	e17	CallerPhone
E18	e18	CallerClues
E19	e19	CallerDevice

The Relationship Tables

The relationship tables save event relationship data and configure the fields that are available to the dml in the external Relationship object, identified by 'R' in the dml.

MDS_RELATION

The MDS_RELATION table is used to save relationship data so that the adapter can continue to process event relationships, and to detect changes in these relationships over a adapter shutdown restart cycle. While not strictly a configuration table, it is described here to clarify the use of other configuration tables. The table has a fixed set of columns that are always in the table, and columns to hold Relationship object field values. The fixed set of columns is:

Column	Type	Description
h_cls	NUMBER	Relation class
h_idx	NUMBER	Relation Index
key_event_cls	NUMBER	Class of key event
key_event_idx	NUMBER	Index of key event
pseudo_dev_cls	NUMBER	Class of pseudo device
pseudo_dev_idx	NUMBER	Index of pseudo device
type	NUMBER	The type of relationship one of: NESTED_OUTAGE(1) MOMENTARY_OUTAGE(2) PARTIAL_RESTORATION(4) RELATED_OUTAGE(8) Pseudo relationships have the same number as their type, with the sign bit (bit 31) set.
active	CHAR(1)	Active flag (Y = active, N = inactive)
when_created	DATE	When the relationship was created
when_completed	DATE	When the relationship was completed or cancelled

The columns that hold the field values can have any name, but they must match the contents of the MDS_RELATION_FIELD table, which is described below. It is suggested that the columns be named similarly to the name of the entity, allowing for the requirements of column names (case insensitivity, etc.). The column types should be VARCHAR2. The columns should be wide enough to hold the data that will be stored in them.

The off-line program that allows dml files to be checked before use can generate suggested contents for the MDS_RELATION and MDS_RELATION_FIELD tables.

MDS_RELATION_EVENT

The MDS_RELATION_EVENT table holds the events that are related to the key events in the MDS_RELATION table. It is not a configuration table. Its columns are:

Column	Type	Description
key_event_cls	NUMBER	Class of key event
key_event_idx	NUMBER	Index of key event
event_cls	NUMBER	Class of related event
event_idx	NUMBER	Index of related event
active	CHAR(1)	Active flag (Y = active, N = inactive)

MDS_ RELATION_FIELD

The MDS_RELATION_FIELD table maps the names of the fields in the Relationship object to the columns used to save their values in the MDS_RELATION table. In addition, it defines the names of all the fields that are available in the Relationship object. If the dml references a field that is not in this table, the adapter will log an error and initialization fails. The columns in the table are:

Column	Type	Description
name	VARCHAR2(32)	The field's name.
col	VARCHAR2(32)	**The column in the MDS_RELATION table that holds the field's value.**

The Code Mapping Tables

The code mapping tables and views are used to translate values in Oracle Utilities Network Management System to and from the equivalent values in the messages to and from the MDS. Typical usages include converting Oracle Utilities Network Management System control zones into dispatch areas on the MDS, and encoding long values in Oracle Utilities Network Management System to shorted encoded values. The dml uses these tables by calling 'mapTableStr' to convert from the MDS value to the Oracle Utilities Network Management System value, and 'mapTableCode' to convert from the Oracle Utilities Network Management System value to the MDS value. A table can be loaded using the function 'loadMapTable'. All these functions take the name of the table as a parameter.

The tables and views have the following columns:

Column	Type	Description
string	VARCHAR2	The value in Oracle Utilities Network Management System.
code	VARCHAR2	The equivalent value in message.

The width of the columns can be any appropriate value.

There are three types of mapping table:

- Those that map from string to code. In these cases the strings must be unique in the table.
- Those that map from code to string. In these cases the code must be unique in the table.
- Those that map both ways. In these cases the strings and codes must be unique.

This can be configured in a map configuration table, described below. If these is no configuration for the table, and informational message is output to the log, and the table is assumed to map both ways.

It is sometimes convenient to add a code column to other Oracle Utilities Network Management System tables to avoid redundant data. Data from these tables can be accessed by use of a view that maps the appropriate column names to 'string' and 'code'.

If a value to be translated is not present in the table, a default value is used. If the default is not specified, the empty string is used.

The types of the tables and their defaults are specified in a map configuration table, which has the following columns:

Column	Type	Description
tablename	VARCHAR2	The name of the table the default applies to.
type	CHAR(1)	The type of the table. 'C' maps from string to code, 'S' maps from code to string, and 'B' maps both ways.
string	VARCHAR2	The default Oracle Utilities Network Management System value.
code	VARCHAR2	The default message value.

The width of the columns can be any appropriate value. A map configuration table is loaded using a function call similar to 'loadMapConfigTable(mds_map_config)'. The configuration document is a convenient place to do this. There can be multiple map configuration tables loaded.

The SRS Message Type Table

The SRS Message Type Table, MDS_SRS_MSG_TYPE, configures the processing of SRSoutput InterSys messages. It has the following columns:

Column	Type	Description
message_type	NUMBER	The SRSoutput message type that this row configures
action	CHAR(1)	How to process it. 'I', 'D', 'P' or 'R'.
proc_relation	CHAR(1)	Should special relationship processing be applied to it? 'Y' or 'N'.
special	CHAR(1)	Should special processing be applied to it? 'Y' or 'N'.
event_cls1	NUMBER	Event class for TRBL_ERT_UPDATE
event_cls2	NUMBER	Event class for TRBL_ERT_UPDATE

The message_type column values are the SRSoutput message types listed in SRSoutput.h.

The action column must be one of these values:

- 'I': completely ignore this message type. Any message type not included in the table is ignored.
- 'D': do not process the message, but apply the proc_relation and special flags if they are set.
- 'P': process the message as is, after applying the flags.
- 'R': reprocess the message's event after applying the flags. Some SRSoutput messages do not hold all of the event's data (e.g., DAMAGE_RPT_UPDATE), and some are used to update the viewer when conditions are to be hidden and may or may not indicate that an event has changed state (e.g., TRBL_REMOVE_ALL). As a consequence, these messages do not hold all the data required to process them. In these cases, the adapter requests SRSService to send all the current data for the message's event.

When set to a value of 'Y', the proc_relation flag instructs the adapter to apply special relation processing to the message. This processing is activated when the appliedRule field of the SRSoutput message contains one of the following values: OUTAGE_PART_REST, OUTAGE_RELATED or OUTAGE_UNRELATED. These indicate that the event is part of a relationship that has been created, deleted or modified. In these cases, the adapter loads the current state of all relationships, reacts to the changes, and reprocesses the message's event after a delay to allow any upcoming changes to the event to arrive before rereading the event's data.

When set to a value of 'Y', the special flag instructs the adapter to apply special, message type specific, processing to the message. The following message types have special processing available:

- TRBL_CLEAR: If the applied rule is not OUTAGE_MERGED, the event is processed by all Output Documents with a 'EventNonexistent trigger.

In addition to the processing of the special flag described above, the following message types are processed differently from normal SRSoutput messages because they are not formatted in the normal manner:

- TRBL_ERT_UPDATE: This message contains a list of estimated time of restoration updates. If this message type is not completely ignored, all the events in the message are reprocessed. As this message contains the events' indexes, but not the events' classes, the potential classes need to be supplied in the event_cls1 and event_cls2 columns. One or both of the classes (normal and momentary event classes) can be configured. If one is zero, it is ignored.
- TRBL_WCB_UPDATE: This message type is not implemented in the adapter. If this message is not configured to be completely ignored, the adapter issues a warning to the log each time this message type is received.

The High Priority Category Table

The High Priority Category Table, MDS_HIGH_PRI_CAT, configures the priorities used by the functions 'highPriTCCategoriesFromClues'. It has the following columns:

Column	Type	Description
group_order	NUMBER	A numeric representation of the trouble code categories
code	VARCHAR2(32)	The trouble codes for each trouble code category
priority	NUMBER	The priority of the trouble codes. Note: The smaller the number the higher the priority. The higher the number the lower the priority

This table assigns the priority of each trouble code within each trouble code category.

Run Time Errors

There are many configuration errors that can be detected at initialization time, causing the adapter to exit. However, some configuration and other errors can only be detected at run-time.

All errors and diagnostic messages are output to the adapter's log file. Important messages are output to the Windows Application Event log. Important errors can be output in XML formatted to an error queue, if an Error Output Document is configured in a Configuration Document. Note that, under some circumstances, it may be impossible to output errors to one or more of these destinations.

The types of run-time errors, and the adapter's reaction to them are:

- Data errors: These errors occur when badly formatted data is received. The input message is discarded.
- Errors detected by the dml: The dml performs appropriate logic. Errors can be logged using the logging functions. Error XML can be sent by triggering the appropriate Output Document.
- dml function call errors: These errors occur when a dml function is called and the prerequisites of the function are not met. The prerequisites of the functions are documented in **DML Function Calls** on page 7-70. This is regarded as a configuration error, causing the adapter to exit.
- Mapping table errors: These errors occur when a map table function is called and the table name supplied by the dml does not exist in the database. This is regarded as a configuration error, causing the adapter to exit.
- Errors when writing to the database: These are initially assumed to be transient, as some tables have constraints that contain time stamps. The adapter pauses for 1 second and re-tries. The adapter will retry 3 times. If all 3 re-tries fail, the adapter exits, on the assumption that there is a severe system problem. If the error occurred while an input message was being processed, and syncpoint is active (configured using the config_MaxBackoutCount field), the message will be re-processed after the adapter restarts. If this cycle is repeated too many times, the message is discarded.
- API call failure: These are treated in the same way as errors writing to the database.

DML Examples

An Output Document

The following is an example of an Output Document that generates a create request document for a small MDS order. Various lines are numbered to allow reference to them in the explanation below. The numbers are not part of the syntax.

```
1   OutputDoc Job
2   queue=$G.OutRequestQueue; associate=O; persist="Y";
3   triggerCreate< SRSoutput; > = isIn($E.status, "ACK", "ASN", "ENR", "RST") &&
                                ($E.outageHdl.cls != $G.momentary) &&
           !findOrder(event, $E.outageHdl);
4   triggerUpdate:S < SRSoutput; > = findOrder(event, $E.outageHdl) &&
                                  (!isIn($E.status, "UNA", "CNL"));
5   {
6   @docType = ChangeJob;
7   if (triggerCreate)
8   {
9      @docType = CreateJob;
10     createOrder();
11  }
12  &RootElement<environment = $G.environment; revision = "1.0.0";> = @docType;
13  &ApplicationArea
14  {
15     &Sender
16     {
17        &Component = NMS;
18        &Confirmation = "Always";
19        &AuthorizationId = "NMS Interface";
20     }
21     &CreationDateTime = formatDTNow();
22     @BODId = getGuid();
23     &BODId = @BODId;
24  }
25  &DataArea
26  {
27     &ExternalNumber = "NMS:"+ $E.outageHdl.idx;
28     # CreationDateTime is outageTime which is
29     # either when the first customer called
30     # or when the device was opened in the model
31     &CreationDateTime=formatDateTime($E.outageTime);
32     &Device:S = $E.devAlias;
33     &Address:S = substring($E.dispAddress, 0, 100);
34     sqlselect @devPhases, @fuseSize, @winter_load,   @summer_load |
              phase_designation, fuse_size, kva_lod_win, kva_lod_sumr |
              classTable($E.devHdl.cls) | "where h_idx = " | $E.devHdl.idx;
35     &DevPhases = @devPhases;
36     &FuseSize = @fuseSize;
37     &WinterLoad = @winter_load;
38     &SummerLoad = @summer_load;
39     &NumCustOut:S = $E.customersOut;
40     &CallerName = $I.0.getCustomerName;
41     &CallerAddr = $I.0.getAddrStreet + “ “ + $I.0.getAddrCity;
42     &CallerPhone = $I.0.getCustomerPhone;
43     &CallerClues = $I.0.getShortDesc;
44     &CallerDevice = $I.0.getDeviceAlias;
45  }
46  $O.BODId = @BODId;
47  }
```

Line 1 declares the document to be an output document named 'Job'. The name is used in diagnostics, and when a document is invoked in a 'triggerOutputDoc' function call (see below).

Line 2 contains three specifications, and illustrates that more than one specification can be placed on one line.

- The first specifies the message queue that the document will be sent out on. The specification assigns the queue to be the value of the external data field named 'OutRequestQueue' in the global configuration object. The queue name is available to the rest of the document in the variable '@queue', and could be changed, if required.
- The second associates the document with the order object. Therefore this document will be used to hold the values of all the elements to be used for change detection when the event is updated with an SRSoutput message.
- The third specifies that the message should be marked as persistent when placed on the queue. Not yet delivered messages are lost when the MQ server stops, if they are not marked as persistent.

Line 3 specifies a trigger, named 'triggerCreate' for the document, which causes an order to be created. It does not have the 'send on change' flag, so change detection is not in effect when this trigger is fired. The trigger specification, between '<' and '>' indicates when the trigger should be evaluated to determine whether it should be fired. In this case the arrival of an asynchronous SRSoutput message causes the trigger to be evaluated. The expression following the '=' is a boolean expression that fires the trigger when it evaluates to true. There may be more than one trigger for an output document. The values of the triggers are available to the rest of the document in variables with the same names as the triggers.

Line 3 also illustrates that a specification can span multiple lines.

Line 4 specifies a trigger, named 'triggerUpdate' for the document, which causes an order to be updated after it has been created. It has the 'send on change' flag, so change detection is in effect when this trigger is fired. The trigger specification, between '<' and '>' indicates when the trigger should be evaluated to determine whether it should be fired. In this case the arrival of an asynchronous SRSoutput message causes the trigger to be evaluated. The expression following the '=' is a boolean expression that fires the trigger when it evaluates to true. There may be more than one trigger for an output document. The values of the triggers are available to the rest of the document in variables with the same names as the triggers.

The '{' on line 5 and the '}' on line 47 enclose the body of the document.

Line 6 sets a default value for the variable @docType, which will be used later to set the root element tag.

Line 7 controls whether lines 9 and 10 are evaluated, depending on which trigger fired. If the triggerCreate trigger fired, 9 and 10 will be evaluated.

Line 9 changes the value of the @docType variable to reflect that the order is new.

Line 10 creates a new order object to save the values of the element's data for change detection.

Line 12 contains the root element tag and attributes. The root element differs from other elements in that the element's tag is specified by the value of the expression to the right hand side of the '=', in this case the value of the variable @docType. The reason for this is so that the same output document can be used for multiple message types, for example to create an order and to update the order. In this example, the 'environment' attribute is set to the value of the external data field named 'environment' in the global configuration object. The root element statement can be anywhere in the output document. All other elements appear in the generated XML in the order that they are within the output document.

Lines 13 to 24 define the 'ApplicationArea' element, with 3 sub-elements, one of which has 3 sub-elements.

The elements on lines 17 to 19 show how constants can be declared. On line 17 the 'NMS' is not surrounded by double quotes ("), because it only contains alphanumeric characters. On line 18 the 'Always' is surrounded by quotes, this acceptable, but not necessary because it only contains alphanumeric characters. On line 19 the 'NMS Interface' must be surrounded by quotes, because it contains a space.

The value of the element on line 21 is supplied by the 'formatDTNow' function that formats the current data and time BOD format CCYY-MM-DDThh:mm:ss.

Line 22 assigns a globally unique id to the variable 'BODId' for use on lines 23 and 46.

Lines 25 to 45 define the 'DataArea' element, with 15 sub-elements.

Line 27 assigns the concatenation of a constant (in quotes) with the event's index.

Lines 28 to 30 are comments and are ignored. Everything between a '#' and the end of a line, inclusive, is a comment and is ignored.

Line 31's element is assigned the time the outage began in the format CCYY-MM-DDThh:mm:ss.

Line 32's element is assigned the alias (name) of the event's interrupting device. Because the 'send on change' flag (S) is present, a change to this element's data will cause the XML to be sent to the MDS.

Line 33's element is assigned the address of the event, truncated to 100 characters. Because the 'send on change' flag (S) is present, a change to this element's data will cause the XML to be sent to the MDS.

Line 34 reads four columns of the interrupting device's facilities (attribute) table. The function 'classTable' supplies the name of the table.

Lines 35 to 38 assigns the values read from the database in line 33 to the appropriate elements.

Line 39's element is set to the number of customers affected by the outage. Because the 'send on change' flag (S) is present, a change to this element's data will cause the XML to be sent to the MDS.

Lines 40 to 44 use the external object 'Incidents' to access customer call data. As there are potentially many incidents associated with an event, the dml has to specify which incident to use. The second component of the name, between the first and second period (.) is the offset into the array of incidents, in this case zero. The last component of the name is the name of the data access method in the Incident class. These examples access the first customer's name, street address, city, phone number, clues, and transformer name. The incidents are normally ordered by the time that they called to report a problem, with the oldest first. If another ordering is required, by total priority for example, the `sort`Incidents() function can be used to change the order.

Line 46 saves the document's globally unique id in the external object for use later in another document.

The XML generated by this output document would look similar to the following:

```
<CreateJob environment="Test" revision="1.0.0">
  <ApplicationArea>
    <Sender>
      <Component>NMS</Component>
      <Confirmation>Always</Confirmation>
      <AuthorizationId>NMS Interface</AuthorizationId>
    </Sender>
    <CreationDateTime>2003-05-01T13:36:13-05:00</CreationDateTime>
    <BODId>guid</BODId>
  </ApplicationArea>
  <DataArea>
   <ExternalNumber>NMS2010</ExternalNumber>
   <CreationDateTime>2003-05-01T13:26:13-05:00</CreationDateTime>
```

```
   <Device>XFM12345678</Device>
   <Address>5800 Yonge St. North York</Address>
   <DevPhases>A</DevPhases>
   <FuseSize>100</FuseSize>
   <WinterLoad>40</WinterLoad>
   <SummerLoad>45</SummerLoad>
   <NumCustOut>4</NumCustOut>
   <CallerName>M.J. McLaughlin</CallerName>
   <CallerAddr>5802 Yonge St. North York</CallerAddr>
   <CallerPhone>416 555-1212</CallerPhone>
   <CallerClues>NC</CallerClues>
   <CallerDevice>XFM12345678</CallerDevice>
 </DataArea>
</CreateJob>
```

An Input Document

The following is an example of an Input Document that processes a document containing completion data for an order. Various lines are numbered to allow reference to them in the explanation below. The numbers are not part of the syntax.

```
1  InputDoc CompletionData
2   queue=$G.InRequestQueue;
3   &RootElement<environment=$G.environment; revision="1.0.0";> =
4      FieldReportSave;
5   BasePath=DataArea;
6   {
7      &JobNumber:R;
8      &Crew
9      {
10        &CrewKey
11        {
12        &CrewName::"MDS";
13        }
14     }
15     &CompletionData
16     {
17        &System:R;
18        &Type:R;
19        &Failure:R;
20        &Cause:R;
21        &InterruptDev:R;
22        &Action:R;
23        &OtherAction::Other;
24        &RestoredTime;
25        &CustomerCaseNotes:::CCN;
26     }
27
28     if (!findOrder(externalId, &JobNumber))
29     {
30        stop;
31     }
32
33     @action=(&CompletionData/Action == "Other")
          ? &CompletionData/OtherAction
          : &CompletionData/Action;
34     @restoredTime = decodeDateTime(&CompletionData/RestoredTime);
35     sqlupdate picklist_info_upd_tr |
          system_om,               type_om,
          failure_om,              cause_om,
```

```
          interrupt_dev_om,             action_text,
          crew_restore |
          &CompletionData/System,  &CompletionData/Type,
          &CompletionData/Failure, &CompletionData/Cause,
          &CompletionData/InterruptDev, @action,
          @restoredTime |
          ref_id | $O.event.idx;

36     # call SRS::setCaseNoteInfo if appropriate
37     if (isSet(&CCN))
38     {
39        setCaseNoteInfo(getCaseNotesForEvent($O.event) + " " +
&CCN);
40     }
41     # set no dtr flag if appropriate
42     if (isIn(&CompletionData/Type, "Customer Trouble",
43        "Other Utilities",
44        "Scheduled/Customer Notified"))
45     {
46        sqlupdate picklist_info_upd_tr | no_dtr_flag | Y |
          ref_id | $O.event.idx;
47     }

       # log change to event
48     @reason_for_update="Completion Information for Job " +
          &JobNumber + " from MDS";
49     sqlinsert picklist_completion_log |
          ref_id,       who,
reason_for_update, when |
          $O.event.idx, &Crew/CrewKey/CrewName,
@reason_for_update, time();
50
51  }
```

Line 1 declares the document to be an input document named ‘CompletionData’.

Line 2 specifies the message queue that the document will be received on. The statement assigns the queue to be the value of the external data field named ‘InRequestQueue’ in the global configuration object.

Line 3 contains the root element tag and attributes. The root element differs from other elements in that the element’s tag is specified by the value of the expression to the right hand side of the ‘=’, in this case ‘FieldReportSave’. The ‘environment’ attribute is set to the value of the external data field named ‘environment’ in the global configuration object. In contrast with output documents, the root element statement is in the header of the input document.

A message arriving on the message queue with the name specified on line 2, and with a root element tag and attributes as specified on line 3, triggers the processing of the message by the input document.

Line 5 specifies the base element’s tag. The base element specifies the sub-element of the root element at which to start processing the elements specified in the body of the input document. The base element is optional, but is convenient in that it reduces the element nesting level in the body of the input document.

The ‘{‘ on line 6 and the ‘}’ on line 51 enclose the body of the document.

Lines 7 to 26 define the tags for the elements that will be processed by the input document.

Line 7 defines a required element ‘JobNumber’. The ‘R’ in the flags field of the element definition indicates that the element must be present in the input XML for the document to be processed. All required elements must be present, otherwise the XML will be ignored.

Line 12 defines an optional element 'CrewName', a sub-element of 'CrewKey', which is in turn a sub-element of 'Crew'. The element has a default value of 'MDS', which is the value used by references to the element in the rest of the document, when the element is not present. If a default value is not present, missing elements have a value of the empty string.

Lines 15 to 26 define the 'CompletionData' element, which has 9 sub-elements, some of which are required.

If all the required elements are present, the data can be processed.

Line 28 calls the 'findOrder' function that finds the order object with an 'externalId' field equal to the JobNumber supplied in the XML. This function returns a boolean indicating whether such an order object was found, or not. This result is inverted by the '!' so that line 30 is executed if 'findOrder' fails. Line 30 is a stop statement, which causes the processing of the document to terminate.

Line 33 sets the variable 'action' to either the contents of the 'OtherAction' element or the 'Action' element, depending on whether 'Action' has a value of 'Other' or not. Both these elements are sub-elements of the 'CompletionData' element. A reference to a sub-element takes the form '&grandparent/parent/subelement'.

Alternatively, an element can be named to make the references more succinct. The element definition of 'CustomerCareNotes' with a name of 'CCN" on line 25, and the references to it on lines 37 and 39 are an example of this.

Line 34 converts the time in the element 'RestoredTime' into a format suitable to be passed to the database, and saves it in the variable 'restoredTime'.

Line 35 saves the values of 5 elements and 2 variables in the 'picklist_info_upd_tr' table in the database using a sqlupdate statement. This table has a key column 'ref_id' that is set to the order's event's index number. If aggregate processing has been configured, and the intention is to save the data in the database for all events in the aggregation, the function 'picklistInfoUpdTr' would be more appropriate.

Line 37 tests whether the element named 'CCN' (the 'CustomerCaseNotes' element in 'CompletionData'), was present in the input XML, using the function 'isSet'. If the element was set line 39 is executed . The function call to 'getCaseNotesForEvent' in the parameter list of the call to 'setCaseNoteInfo', calls an Oracle Utilities Network Management System API to read the current value of the order's event case notes. The value returned is concatenated with a space and the value of the CCN element. The resulting value is saved to the event's case notes using the function 'setCaseNoteInfo'.

Lines 42 to 47 set the 'no_dtr_flag' column in the 'picklist_info_upd_tr' table to 'Y', if the 'CompletionData/Type' element indicates that the trouble was not with the utility's equipment. The function 'isIn' returns true if its first parameter matches any one of the rest of its parameters, false otherwise

Lines 48 and 49 logs the change to the event by inserting a row into the picklist_completion_log table. If aggregate processing has been configured, and the intention is to save the data in the database for all events in the aggregation, the function 'picklistCompLog' would be more appropriate.

The following XML would be processed by the example input document

```
<FieldReportSave environment="Test" revision="1.0.0">
 <ApplicationArea>
   <Sender>
     <Component>MDS</Component>
     <Confirmation>Never</Confirmation>
     <AuthorizationId>MDS Interface</AuthorizationId>
   </Sender>
   <CreationDateTime>2003-05-01T14:30:17-05:00</CreationDateTime>
   <BODId>guid</BODId>
 </ApplicationArea>
 <DataArea>
  <JobNumber>MDS7704</JobNumber>
  <CompletionData>
   <System>4KV</System>
   <Type>Lateral</Type>
   <Failure>Fuse</Failure>
   <Cause>Animal</Cause>
   <InterruptDev>XFM12345678</InterruptDev>
   <Action>Fuse replaced</Action>
   <CustomerCaseNotes>Temporary Repair: Scheduled for Tuesday</
CustomerCaseNotes>
  </CompletionData>
 </DataArea>
</FieldReportSave>
```

DML Reference

This section contains a full reference for the dynamic message language used in the Generic WebSphere MQ Mobile Adapter configuration dml files.

Lexical Conventions

A dml configuration consists of one or more files. Each file is processed in turn to generate a sequence of tokens, which are further processed into internal data structures used at run time to generate XML documents and process XML documents.

Syntax notation

In the syntax notation used in this Appendix, syntactic categories are indicated by *italic* type, and literal words and characters in **bold** type. An optional part of the syntax is indicated by enclosing it in square brackets ([]). An ellipsis (…) indicates that the preceding part of the syntax can be optionally repeated an arbitrary number of times.

Tokens

There are seven kinds of tokens: names, strings, quoted strings, keywords, operators, and other separators. Blanks, tabs, line feeds, carriage returns, and comments (described below), are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent tokens.

Comments

The character # starts a comment which terminates at the end of the line on which it occurs.

Include Directive

The include directive can appear anywhere in an input file and takes the form:

```
include file name
```

The contents of the *file name* are processed as if they replaced the include directive itself.

Keywords

The following names are reserved for use as keywords and may not be used otherwise:

```
associate persist BasePath classTable ConfigDoc else elseif for if
include InputDoc OutputDoc queue RootElement sortIncidents sqlinsert
sqlselect sqlupdate static stop VERSION
```

The following characters are used as operators or for punctuation:

```
@ $ & / ! | = : , " ; . ( ) [ ] < > { } +
```

The following character combinations are used as operators:

```
== != && ||
```

Names

A *name* is an arbitrarily long sequence of letters and digits. The first character must be a letter. The underscore _ counts as a letter. Upper and lower case letters are different. All characters are significant.

Strings

A *string* is an arbitrarily long sequence of letters and digits. . The underscore _ counts as a letter. Upper and lower case letters are different. All characters are significant.

Quoted Strings

A *quoted string* is an arbitrarily long sequence of characters, enclosed in double quotes "". To represent the double quote character in a quoted string, use two double quotes "".

Constants

A constant is one of:

- name
- string
- quoted string

Version Directive

The version directive takes the form:

```
VERSION = string;
```

The *VERSION* directive serves to identify the version of the current dml file and must appear outside any document definitions. When this directive is encountered, the current file name and the supplied version *string* are output to the log.

Basic Concepts

Type(s)

DML is a typeless language, or perhaps more accurately, a singly typed language. All values consist of character strings. DML has no concept of numbers. To illustrate this, *strings* can be used to pass positive integers to *functions*. For example:

```
@truncatedString = substring(@address, 0, 100);
```

However, *quoted strings* must be used to pass negative integers. For example:

```
@lastCharacter = substring(@characters, "-1", 1);
```

It should be emphasized that it is the *function*, not the dml that is interpreting the character string as an integer.

In certain contexts, a value is used as a *boolean*, or logical, value. Any non-empty string is considered *true*, and the empty string (written as "") is considered *false*. A *boolean expression* or *function* returns "Y" when *true* and "" when *false*.

Definitions and References

There are a number of value bearing *entities* in dml. Apart from one exception (*variables*), they must be defined before they are referenced. The definition introduces the *entity*, and defines certain modifiers to the *entity*. An *entity* is referenced when its value is used in an *expression* or when it is assigned to in an assignment *statement*. When a *variable* uses the default set of modifiers, it can be implicitly defined when first assigned.

Entities

The *entities* in a document carry values and have modifiers that change their behavior.

Variables

A *variable* is used to save intermediate results in a *document*. A *variable definition* takes the form:

```
@name:modifiers
```

The name must be unique amongst the *variables* within the document. Other entities can have the same name. The **:***modifiers* is optional.

If the **:***modifiers* are not required, variables can be implicitly defined when they are assigned to.

A *variable reference* takes the form:

```
@name
```

Variables can be set and referenced in all document types.

Elements

An *element* generates an XML element in Output Documents, and supplies an input XML element value or sub-elements in Input Documents. There are to kinds of *element* definitions: plain *elements* and array *elements*.

plain element definitions and an *array element definitions* are known collectively as *element definitions*.

Plain Element Definitions

Plain elements are referred to as *elements* in the rest of this section.

An *element definition* takes the form:

```
&name:modifiers < attribute definitions >
```

The name of an *element* is its XML tag, and must be unique amongst the *elements* within its immediately enclosing *element* or document. Other entities can have the same name. The **:***modifiers* and *attribute definitions* are optional. The format of *attribute definitions* is described below in section **Attributes** on page 7-60. *Elements* must either have values but no sub-elements or must have sub-elements but no value. (This may be changed in a future version of the adapter).

Array Element Definitions

Array elements can only appear in Input Documents, and are used when more that one element with the same tag can be sub-elements of the same element in the input XML. An *array element definition* takes one of two forms: with an unspecified index, and with a specified index attribute.

The unspecified index form is:

```
&name[]:modifiers < attribute definitions >
```

The specified index form is:

```
&name[name]:modifiers < attribute definitions > R(constant list)
```

where the name in square brackets is the name of the index attribute, and R(*constant list*) is the required index list, which is optional. The required index list defines the values of the index attribute that must be in the input XML before the enclosing Input Document will be used to process the input XML.

Array *element modifiers* can only contain an alternate name for the array *element*.

The name of an array *element* is its XML tag, and must be unique amongst the *elements* within its immediately enclosing *element* or document. Other entities can have the same name. The **:***modifiers* and *attribute definitions* are optional. The format of *attribute definitions* is described below in section **Attributes** on page 7-60. Array *Elements* must either have values but no sub-elements or must have sub-elements but no value.

Element References

The values of plain *elements* with values in Input Documents can be obtained using a *plain element reference* taking the form:

```
&element identifier
```

The *element identifier* has two forms:

The *full element path*, which is either the *name* of the element, if it is not a sub-element of another element, or the full element path of the sub-element's enclosing element, followed by a slash (/) followed by the element's *name*.

The element's alternate name, which is defined in the element's *modifiers*.

If the element has an alternate name, the full element path cannot be used.

The values of individual *elements* in an array *element* with values can be obtained using an *array element reference* taking the form:

```
&element identifier[index value]
```

The index value is either the value of the index attribute, in the case of a specified index, or a number in the case of an unspecified index, the elements being numbered in the order that the elements were in the input XML, starting at zero.

plain element references and an *array element references* are known collectively as *element references*.

Elements can only be referenced in Input Documents.

Attributes

An *attribute* generates an XML element attribute in Output Documents, and supplies an input XML element attribute value in Input Documents. An *attribute* definition is part of an *element's attribute definitions* and takes the form:

```
name:modifiers
```

The name of an *attribute* is its XML attribute, and must be unique amongst the *attributes* within its immediately enclosing *element* or document. Other entities can have the same name. The **:***modifiers* are optional.

In Output Documents *attributes* must have their values assigned to them where they are defined in the form:

```
NAME:modifiers = expression;
```

The definition of an *attribute* in an Input Document takes the form:

```
NAME:modifiers;
```

Attributes are defined as part of the *attribute definitions* of their *element* as described above in section These *attribute definitions* are one or more *attribute* definitions.

The values of an *attribute* can be obtained using an *attribute* reference taking the form:

```
element reference<name>
```

Attributes can only be referenced in Input Documents.

Entity Modifiers

There are three entity *modifiers* they are:

- The *flags* which modify the behavior of the entity. See section **Flags** on page 7-23 in the main document for the uses of flags.
- The *default value*, which is used when entity is referenced but has no value. See section **Defaults** on page 7-23 in the main document for the circumstances that the default is used.
- The *alternate name*, which is used to give the entity an alternate name. See section **Alternate Names** on page 7-24 in the main document for the circumstances that the alternate name is used.

The *modifiers* are defined in the order *flags*, *default value*, *alternate name* and take the form:

```
:constant:constant:name
```

If a *modifier* at the end of the *modifiers* is empty the colon (:) must not be present. As a consequence, if all modifiers are empty, the *modifiers* are not present

External Data

External data is available from the various external data objects, described above in section **The External Objects** on page 7-27. All fields in the objects can be read, but some objects or individual fields are read-only, i.e. they cannot be written.

A reference to an external object field takes the form:

```
$external data object identifier.field name
```

where the *external data object identifier* is the letter associated with the object and the *field name* is the *name* of the field (note that they are separated by a period (.)). Some fields (for example Handles) have sub-fields. A reference to a sub-field takes the form:

```
$external data object identifier.field name.sub-field name
```

Most external objects have one instance at a time, but the Incident Object can have zero, one or more, depending on the number of incidents that have grouped to the current event. To reference an individual incident field, an offset to the incident into the array of incidents is required. This offset starts at zero, and is a *constant* which must only contain digits, known as an *offset*. The normal order of incidents is the order that they were received by Oracle Utilities Network Management System. This order can be altered by use of the sortIncidents function. A reference to an incident field takes the form:

```
$I.offset.field name
```

Incident references can contain *sub-fields*.
These are known as *external field references*.

Note that some external object fields are read-only, i.e. they cannot be set by the dml. The read-only status of each external object is listed below:

- The Order Object ('O'): All the fields listed in section **Permanent Order Object Fields** on page 7-135, **Permanent Order Object Fields** on page 7-135 of **DML Function Calls** on page 7-70 are read-only. All other fields are read/write.
- The Relationship Object ('R'): All the fields listed in section **Permanent Relationship Object Fields** on page 7-135, **Permanent Relationship Object Fields** on page 7-135 of **DML Function Calls** on page 7-70 are read-only. All other fields are read/write.
- The Event Object ('E'): All fields are read-only.
- The Incident Object('I'): All fields are read-only.
- The Global Data Object ('G'): All fields are read/write.
- The Trigger Parameter Object ('T'): All fields are read/write.

Functions

A *function* is called using the following form:

```
name([paramter1] [, paramter2]  …)
```

Where *name* is the name of the function and parmeter2, parameter2, … are *expressions*. All functions return a value, but in some cases the value is always the empty string, implying that they are only called for their side effects. The functions available and their parameters are described in **DML Function Calls** on page 7-70.

Expressions

An expression is a combination of dml components that yield a value. Expressions are combined using operators. The following table shows the expressions and the operators:

Expression	Value	Notes
constant	The constant	
variable reference	The variable's current value	
element reference	The element's current value	
attribute reference	The attribute's current value	
external field reference	The field's current value	
function	The function's return value	
expression1 **+** *expression2*	The concatenation of the two *expressions*	
(*expression***)**	The expression's value	Used to alter the precedence of operators.
expression1 **?** *expression2* **:** *expression3*	If *expression1* is *true*, *expression2*. If *expression1* is *false*, *expression3*.	Logical alternation. See note below.
! *expression*	If *expression* is *true*, *false*. If *expression* is *false*, *true*.	Logical NOT.
expression1 **&&** *expression2*	*true* if both *expression1* and *expression2* are *true*, *false* otherwise	Logical AND. See note below.

Expression	Value	Notes
expression1 || *expression2*	*true* if either *expression1* or *expression2* is *true*, *false* otherwise	Logical OR. See note below.
expression1 **==** *expression2*	*true* if *expression1* is an exact duplicate of *expression2* is *true*, *false* otherwise	
expression1 **!=** *expression2*	*true* if *expression1* is not an exact duplicate of *expression2* is *true*, *false* otherwise	

Note: The logical alternation, AND, and OR expressions are evaluated left to right and expressions that do not need to be evaluated are not evaluated, and any side effects (e.g., due to a function call) do not occur. Specifically:

Alternation: *expression1* is evaluated. If it is *true* only *expression2* is evaluated, otherwise only *expression3* is evaluated.

AND: *expression1* is evaluated. If it is *false*, *expression2* is not evaluated.

OR: *expression1* is evaluated. If it is *true*, *expression2* is not evaluated.

Lists

Name List

A *name list* is one or more *names* separated by commas (**,**). For example:

```
h_cls, h_idx
```

Constant List

A *constant list* is one or more *constants*, separated by commas (**,**). For example:

None, "$%I99", 13, "-3", "This is a ""quoted"" string"

Variable Reference Lists

A *variable reference list* is one or more *variable references*, separated by commas (**,**). For example:

```
@devPhases, @pole_number, @winter_load, @summer_load
```

Expression List

A *expression list* is one or more *expressions*, separated by commas (**,**). For example:

```
@a + @b, formatDateTime(@time), $E.outageHdl.idx, none
```

Statements

Statements are the basic processing units in dml. Many of them are terminated using the *statement terminator*, **;** (semi-colon).

Statement Blocks

A number of statements require one or more *statements* grouped together. This is achieved using a *statement block*, which takes the form:

```
{
   statements
}
```

Where *statements* is one or more *statements*.

Variable Assignment Statement

This statement assigns a value to a variable and takes the form:

```
variable reference = expression;
```

Element Definition Statement

Element definitions differ between elements with values (simple elements) and those with sub-elements (compound elements). They also differ between those in Output Documents (output elements) and those in Input Documents (input elements). The four flavors are described in the following sections.

Output Element Definition Statements

These statements are only valid in Output Documents.

Simple Output Element Definition Statement

This statement assigns a value to an element and takes the form:

```
element definition = expression;
```

Compound Output Element Definition Statement

This statement defines an element with one or more sub-elements and takes the form:

```
element definition
statement block
```

The statement block must contain at least one output element definition statement, but can also contain other statements allowed in Output Documents.

Input Element Definition Statements

These statements are only valid in Input Documents.

Simple Input Element Definition Statement

This statement defines an element that can accept a value from incoming XML documents and takes the form:

```
element definition;
```

Compound Input Element Definition Statement

This statement defines an element with sub-elements that can accept values value from incoming XML documents and takes the form:

```
element definition
statement block
```

The statement block can only contain input element definition statements and must contain at least one.

External Data Assignment Statement

This statement assigns a value to an external object field and takes the form:

```
external field reference = expression;
```

Function Statement

This statement is used to call a function for its side effects and the return value is either the empty string or can be ignored. It takes the form:

```
function expression;
```

SQL Select Statement

This statement is used to read data from the database and takes the form:

```
sqlselect variable reference list | name list | [static] expression [ |
expression …] ;
```

SQL Insert Statements

This statement is used to insert a row of data in a database table and takes the form:

```
sqlinsert expression | name list | expression list ;
```

SQL Update Statements

This statement is used to insert or update a row of data in a database table and takes the form:

```
sqlupdate expression | name list | expression list | name list |
expression list ;
```

If Statement

This statement is used to alter the flow of expression evaluation, and output element selection, based on the value of an expression. It takes the form:

```
if (expression)
   statement block 1
else if (expression)
   statement block 2
else
   statement block 3
```

elseif is a synonym for **else if**.

There can be any number of **else if**'s, including none. The **else** is optional.

If the *expression* of the **if** is *true*, it's *statement block* (1 in this example) is evaluated.

Otherwise, if the *expression* of the first **else if**, if any, is *true*, it's *statement block* (2 in this example) is evaluated.

Otherwise, if the *expression* of the next **else if**, if any, is *true*, it's *statement block* is evaluated.

Otherwise, the **else's** *statement block* (3 in this example) is evaluated.

A maximum of one *statement block* will be evaluated in any *if statement*.

For Statement

This statement is use to iterate through all elements in an array element. It can only appear in an Input Document. It takes the form:

```
for (&element identifier[], variable reference)
statement block
```

Stop Statement

This statement is used to terminate processing of a document. In addition, it prevents an Output Document from sending its XML. It takes the form:

```
stop;
```

Root Element Statement

This statement is used to generate an Output Document's root element and to select the Input Document or Documents that the input XML can be processed by. In an Output Document it is located in the document's *statement block*. In an Input Document it is located in the *input document header*. It takes the form:

```
&RootElement [ < attribute definitions > ] = expression ;
```

Documents

Output Document Header

The *output document header* consists of the following specifications, in any order.

Queue Specification

This specification specifies the queue to which the Output Document's XML is directed to, and takes the form:

```
queue = expression ;
```

The queue is available to the Output Document's statement block as a variable named queue, and can be assigned to in the following manner:

```
@queue = expression ;
```

The queue specification is optional, and defaults to the empty string. If the queue specification is defaulted, it must be set in the Output Document's statement block. There may only be a maximum of one queue specification in the header of each Output Document.

Trigger Specification

An Output Document must have at least one trigger specification, and can have an arbitrary number greater than one.

Association Specification

This specification associates the Output Document with an external object, and takes the form:

```
associate = external data object identifier ;
```

The only *external data object identifier* currently supported is **O**, the Order Object. One and only one Output Document must be associated with the Order Object.

Persistence Specification

This specification allows you to set the persistence flag for outgoing messages, and takes the form:

```
persist = "Y" or "N";
```

If this specification is omitted, the message will have the default persistence setting configured for the queue it is being placed on.

Output Document

This document is used to generate XML and send it to the MDS. It takes the form:

```
OutputDoc name
output document header
statement block
```

Input Document Header

The *input document header* consists of one and only one root element statement and the following specifications, in any order.

Queue Specification

This specification specifies the queue on which the Input Document's XML is received from, and takes the form:

```
queue = expression ;
```

The queue is available to the Input Document's statement block as a variable named queue.

There must be one and only one queue specification in the header of each Input Document.

Base Path Specification

The specification specifies the sub-element of the root element of the XML document which contains all elements that will be processed by the Input Document. It takes the form:

```
BasePath = full element path;
```

Input Document

This document is used to process input XML from the MDS. It takes the form:

```
InputDoc name
input document header
statement block
```

Configuration Document

This document is used to set configuration data and load configuration tables from the database at initialization time. It takes the form:

```
ConfigDoc name
statement block
```

Order of Document Processing and Other Considerations

The dml is read during adapter initialization in the order that the files are specified in the command line. Files from the command line must contain only complete documents, but files read using the **include** directive can contain any valid dml fragment, dependent on the context of the directive.

The processing of statements within a document is strictly from top to bottom in the order the statements were read during initialization, except when altered by a control flow statement (e.g., **if** and **for** statements). Statements in a document are processed until one of the following situations occur:

The last statement in the document is reached. If the document is an Output Document, the resulting XML is delivered to the queue specified in the document's queue specification.

A **stop** statement is processed. If the document is a Configuration Document, the adapter sends an error message to the log and then exits.

A run time configuration error occurs. In all cases, the adapter sends an error message to the log and then exits.

An unrecoverable run time error occurs (e.g., DBService is not available to read or write a database table). In all cases, the adapter sends an error message to the log and then exits.

The processing of specifications in a document header is not necessarily in top to bottom order. Each document is described below.

Output Documents

Output Document specifications are processed in the following order:

The association specification is processed once during initialization.

The queue specification is evaluated just before the first statement of the document is processed.

When a trigger event occurs, all Output Documents are examined in top to bottom order to determine whether the event should trigger each document.

If the trigger event's type matches at least one trigger specification in the document, the triggers of that type are processed in top to bottom order, until one evaluates to *true*. In this case, all other triggers have their value set to *false*, even if they have not been processed. If all triggers evaluate to *false* the document is not processed due to the trigger event.

If the Output Document is triggered, it is fully processed before the next Output Document is examined in order to determine whether the event should trigger the next document.

Input Documents

All Input Document header specifications are evaluated at initialization, in the order base path specification, queue specification, and then root element specification.

When an input XML document arrives, each Input Document is examined in top to bottom order to determine whether the XML satisfies the root element specification. If no document matches the incoming XML, the XML is discarded. If at least one Input Document is eligible, the elements in the XML are delivered to the documents in the order they appear in the XML document. Once all elements have been delivered, each document is examined in top to bottom order to determine whether all required elements are present, and are processed in top to bottom order if the elements are present. Once all Input Documents have been processed, the XML is discarded.

Configuration Documents

Configuration Documents have no header specifications. All Configuration Documents are processed in top to bottom order. Once they all have been processed successfully, they are discarded.

Ordering of Incidents in the Incident Object

When processing starts for a document, the incidents in the Incident Object are in their normal order, i.e. the order that they were received by Oracle Utilities Network Management System. This order can be changed during the processing of the document by calling the sortIncidents function. The ordering remains the same during the processing, unless sortIncidents is called again. The order is set back to normal when the document processing finishes.

Interactions between Threads

The adapter is a multi-threaded process. Therefore more than one document can be processed at the same time, increasing performance. There is at least one thread for Output Document processing, and at least one thread for each Input Document queue. More threads can be configured using a Configuration Document. The adapter uses a number of other threads for internal processing.

This has a number of implications.

While trigger events are queued internally in the order that they occur, and are extracted from this queue in the order that they were queued, there is no guarantee that the Output Documents triggered by these events will complete their processing in the order the events were queued. This means the XML messages may be delivered to the MQSeries queue in an unexpected order. If this behavior is inappropriate, it can be eliminated at the expense of performance by using only one output thread, and by setting the config_Event_QueueDelay configuration parameter to zero.

A similar situation exists with input XML documents. They also cannot be guaranteed to update Oracle Utilities Network Management System in the order that they arrive. This situation can be improved at the expense of performance by limiting each input queue to one thread. It may be possible to eliminate it completely if the interface can be configured to use only one input queue.

Note that trigger events and input XML that affect particular Oracle Utilities Network Management System events, Order Objects, and Relationship Objects are processed in the order that they are queued. These situations are discussed below.

Note, however, that there is an inherent race condition in loosely coupled interfaces (the type implemented by the adapter) that use messages to communicate. Events can occur in Oracle Utilities Network Management System and the MDS that alter the state of Oracle Utilities Network Management System events and MDS orders almost simultaneously and it cannot be predicted whether the change on one system affects the other system first, or vice versa. Paradoxically, this situation can be improved by increasing the rate at which messages are processed, i.e. by increasing the number of threads.

The input, output, and internal threads need to coordinate access to various shared resources. Most of this coordination is invisible on the dml level, but a number of aspects of the coordination are worth consideration when writing dml.

One specific means of coordination is known as a mutex (for mutual exclusion). To access a shared resource that is protected by a mutex, the thread requests the mutex. If the mutex is free, the thread acquires the mutex. If another thread requests the mutex, it blocks (is suspended) until it is free. When the original thread has finished accessing the shared resource, it releases the mutex, making it free. The release unblocks one thread waiting for the mutex. When there is the potential for more than one mutex to be in use, there is a danger of a deadlock if one thread enters the mutexes in a different order from another thread. One a deadlock occurs, neither thread will ever run again.

Each thread runs in a separate environment, with a copy of each relevant document, but needs to share a number of resources. The access to these objects must be properly coordinated to prevent inconsistent access due to multiple threads updating the resource at the same time. These resources and the mechanisms used to prevent inconsistent access are:

- The Global Data Object: The update of a single field is atomic (the update will be complete before any other thread can attempt to read or update the field). There is no coordination of updates to multiple fields. To avoid this problem, use a single field.
- The Order Object: There is one Order Object for each order that has been created. Access to an individual Order Object is coordinated so that only one document can access the Order Object at one time. When a document in a thread needs to access an Order Object it calls findOrder. When successful, findOrder acquires the order's mutex, preventing any other thread from accessing the Order Object. The order's mutex is automatically acquired when the order is created by calling createOrder. The order's mutex is automatically released in the following cases: when the document processing is terminated, and when findOrder or createOrder is called. This is to prevent deadlocks. A consequence of this is that if a trigger event triggers two Output Documents needing the same order, or two Input Documents are triggered by one input XML message, the state of the order when the second document starts processing is not guaranteed to be the same, because another thread may have altered its state.
- The Relationship Object: There is one Relationship Object for each relationship that has been created. Access to an individual Relationship Object is coordinated so that only one document can access the Relationship Object at one time. When a document in a thread needs to access a Relationship Object it calls findRelation. When successful, findRelation acquires the relationship's mutex, preventing any other thread from accessing the Relationship Object. The relationship's mutex is automatically acquired when the relationship is created by calling createRelation. The relationship's mutex is automatically released in the following cases: when the document processing is terminated, and when findRelation or createRelation is called. This is to prevent deadlocks. A consequence of this is that if a trigger event triggers two Output Documents using needing the same relationship, or two Input Documents are triggered by one input XML message, the state of the relationship when the second document starts processing is not guaranteed to be the same, because another thread may have altered its state. In addition, when findOrder or createOrder is called, the mutex for the order's relationship is automatically acquired. This has the consequence that calling findOrder or createOrder after calling findRelation causes a configuration error if the order is not in the relationship. Similarly, calling findRelation after calling findOrder or createOrder causes a configuration error if the order is not in the relationship. This behavior is necessary to prevent deadlocks.
- The current event: SRSoutput message for the same event must be processed in the order that they were sent by SRSService, otherwise the MDS will not receive up to date data. This prevented by a mutex that is acquired before the SRSoutput message is processed, and released automatically when it has been processed.

DML Function Calls

The importance of meeting the prerequisite specifications for all functions cannot be emphasized too much. If a prerequisite is not met, the adapter will exit with a fatal error message. Some of the prerequisites are written is short form because they are so common. An explanation of how to meet these prerequisites follows:

- The current crew is set: The current crew is set by a successful call to the createCrew function, and by any of the findCrew… functions.
- The current order is set: the current order is set by the createOrder and findOrder functions.
- The current event is set: the current event is set when the current order is set, when an OutputDocument has been triggered by the arrival of an SRSoutput message, and when a call to findEventObject is successful.
- The current damage report is set : the current damage report is set by a call to findOrCreateDamage

In addition, access to the E (event), I (incident), O (order) and R (relation) objects will only work if their prerequisites are met. If these are not met, the adapter does not exit, but all accesses fail, meaning no data can be read or written. Their prerequisites are:

- E: the current event is set.
- I: the current event is set (and it has incidents)
- O: the current order is set.
- R: the current relation is set by a call to findRelation.

List of Functions

Crew Functions

These functions create and update crews.

createCrew

NAME:

createCrew Create a new crew.

SYNOPSIS:

createCrew(crew ID, crew field, data, [crew field, data], ..)

PARAMETERS:

crew ID	The crew ID
crew field	The crew field to update.
crewType	calls Crew::crewType() API
zoneName	calls Crew::zoneName() API
crewSize	calls Crew::crewSize() API
contact	calls Crew::contact() API
crewId	calls Crew::crewId() API
mobileNum	calls Crew::mobileNum() API
pagerNum	calls Crew::pagerNum() API
zoneHdl	calls Crew::zoneHdl() API
crewCategory	calls Crew::crewCategory() API
crewCenter	calls Crew::crewCenter() API
crewGroup	calls Crew::crewGroup() API
crewSupervisor	calls Crew::crewSupervisor() API
externalKey	calls Crew::externalKey() API
data	The data written to the crew field

Note: If the Crew Icons Window is used make sure that the following crew fields: crewType, contact, zoneName, zoneHdl, crewCategory, crewCenter, crewSupervisor, are included otherwise the new crew will not appear in the Window.PRE-REQUISITE:

The crew field parameters are valid.

DESCRIPTION:

Create a new crew with the specified information using the Crew::createCrew API. If successful, set the current crew to the new crew.

RETURN VALUE:

True when successful.

False when unsuccessful.

A crew with the specified crew ID exists. This can be determined by a call to findCrewById().

The Crew::createCrew() API call fails.

Note that the current crew is not set if the call fails.

DIAGNOSTICS:

Error messages are output to the error log.

findCrewById

NAME:

findCrewById Find an active or inactive crew based on its crew ID SYNOPSIS:

findCrewById(crew ID)

crew ID The crew ID.

PRE-REQUISITE:

None.

DESCRIPTION:

Find the crew with the specified crew ID.

RETURN VALUE:

True when successful. The current crew is set to the crew that was found.

False when unsuccessful.DIAGNOSTICS:

None.

findCrewByExternalKey

NAME:

findCrewByExternalKey Find an active or inactive crew based on its external key

SYNOPSIS:

findCrewByExternalKey(externalKey)

externalKey The external key.

PRE-REQUISITE:

None.

DESCRIPTION:

Find the crew with the specified external key.

RETURN VALUE:

True, when successful. The current crew is set to the crew that was found.

False when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

findCrewByIdSubStr

NAME:

findCrewByIdSubStr   Find an active or inactive crew based on a substring of the crew ID

SYNOPSIS:

findCrewByIdSubStr(string)

string   A string.

PRE-REQUISITE:

None.

DESCRIPTION:

Find a crew whose crew ID contains the string.

RETURN VALUE:

True when successful. The current crew is set to the crew that was found.

False when unsuccessful.

DIAGNOSTICS:

None.

findCrewByExtKeySubStr

NAME:

findCrewByExtKeySubStr   Find an active or inactive crew based on a substring of its external key

SYNOPSIS:

findCrewByExternalKey(string)

string   A string.PRE-REQUISITE:

None.

DESCRIPTION:

Find a crew whose external key contains the string.RETURN VALUE:

True, when successful. The current crew is set to the crew that was found.

False when unsuccessful.DIAGNOSTICS:

Error messages are output to the error log.

updateCrew

NAME:

updateCrew Update the current crew's information.

SYNOPSIS:

updateCrew(crew field, data, [crew field, data], ..)

PARAMETERS:

crew field	The crew's field to update.
crewType	calls Crew::crewType() API
zoneName	calls Crew::zoneName() API
crewSize	calls Crew::crewSize() API
contact	calls Crew::contact() API
crewId	calls Crew::crewId() API
mobileNum	calls Crew::mobileNum() API
pagerNum	calls Crew::pagerNum() API
zoneHdl	calls Crew::zoneHdl() API
crewCategory	calls Crew::crewCategory() API
crewCenter	calls Crew::crewCenter() API
crewGroup	calls Crew::crewGroup() API
crewSupervisor	calls Crew::crewSupervisor() API
externalKey	calls Crew::externalKey() API
data	The data written to the crew field

PRE-REQUISITE:

The current crew is set.

The crew field parameters are valid.

DESCRIPTION:

Update the current crew's crew field with data and invoke the Crew::commit() API to commit the changes.

RETURN VALUE:

True when successful.

False when unsuccessful.

The Crew::commit() API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

dispatchCrew

NAME:

dispatchCrew	Dispatch the current crew to the current order

SYNOPSIS:

dispatchCrew()

PARAMETERS:

None.

PRE-REQUISITE:

The current crew is set.

The current order is set.

DESCRIPTION:

If the crew is already dispatched to another event, the previous dispatch is changed to an assignment.

Invoke the Crew::dispatch() API for the order's active event. If the order is aggregated, assign the crew to the other events.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

arriveCrew

NAME:

arriveCrew	Update the current crew's information to reflect that the crew has arrived on site for an order.

SYNOPSIS:

arriveCrew()

arriveCrew(<u>time</u>)

PARAMETERS:

<u>time</u>	The time the crew arrived in internal format. If this parameter is not supplied, the current time is used.

PRE-REQUISITE:

The current crew is set.

The current order is set.

DESCRIPTION:

If the crew is not dispatched to the order, the crew is dispatched to the order, using the logic described in the description of the 'dispatchCrew' function.

If the time is zero or invalid it is set to the current time

Invoke the CrewDispatch::arrived() API for the order's active event.

RETURN VALUE:

The empty string.DIAGNOSTICS:

Error messages are output to the error log.

assignCrew

NAME:

assignCrew Assign the current crew to the current order

SYNOPSIS:

assignCrew()

PARAMETERS:

None.

PRE-REQUISITE:

The current crew is set.

The current order is set.

DESCRIPTION:

If the crew is dispatched to the order, undispatch it.

Invoke the Crew::assign() API for all events associated with the order.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

unassignCrew

NAME:

unassignCrew Unassign the current crew from the current order

SYNOPSIS:

unassignCrew()

PARAMETERS:

None.

PRE-REQUISITE:

The current crew is set.

The current order is set.

DESCRIPTION:

If the crew is assigned or dispatched to any events associated with the order, invoke the Crew::unassign() API or Crew::undispatch API, respectively, for the events.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

activateCrew

NAME:

activateCrew Change the state of the current crew to active or inactive.

SYNOPSIS:

activateCrew(state)

PARAMETERS:

state The state of the crew.

Y Activate the current crew.

N Deactivate the current crew, set it off-shift, and remove all crew assignments and jobs.

PRE-REQUISITE:

The current crew is set.

DESCRIPTION:

Invoke Crew::setActivation() API.

RETURN VALUE:

True when successful.

False when unsuccessful.

The assignments and jobs cannot be removed from the current crew

The Crew::setActivation() API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

availableCrew

NAME:

availableCrew Change the availability of the current crew.

SYNOPSIS:

availableCrew(state)

PARAMETERS:

state The availability of the crew.

Y Make the current crew available.

N Make the current crew unavailable; remove all crew assignments and jobs.

PRE-REQUISITE:

The current crew is set.

DESCRIPTION:

Invoke Crew::setAvailability() API.

RETURN VALUE:

True when successful.

False when unsuccessful.

The assignments and jobs cannot be removed from the current crew

The Crew::setAvailability() API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

releaseCrews

NAME:

releaseCrews Release all crews from the current order.

SYNOPSIS:

releaseCrews

PARAMETERS:

None.

PRE-REQUISITE:

Current order is set.

DESCRIPTION:

Undispatch and unassign all crews related to the current order.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

crewActive

NAME:

crewActive Determine if the current crew is active.

SYNOPSIS:

crewActive()

PARAMETERS:

None

PRE-REQUISITE:

The current crew is set.

DESCRIPTION:

Invoke Crew::isActive() API.

RETURN VALUE:

True, when the current crew is active.

False, when current crew is not active.

DIAGNOSTICS:

Error messages are output to the error log.

setCrewOnShift

NAME:

setCrewOnShift Change the state of the current crew to on shfit or off shift.

SYNOPSIS:

setCrewOnShift(onShift, time)

PARAMETERS:

onShift	The state of the crew
Y	Set the current crew to on-shift. This will also activate the crew if it is currently inactive.
N	Set the current crew to off-shift and suspend any jobs.
time	The time the crew shift change occurred. If this parameter is not supplied, the current time is used.

PRE-REQUISITE:

The current crew is set.

DESCRIPTION:

Invoke Crew::setOnShift() API.

RETURN VALUE:

True when successful.

False when unsuccessful.

The Crew::setOnShift() API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

crewOnShift

NAME:

crewOnShift Determine if the current crew is on shift.

SYNOPSIS:

crewOnShift()

PARAMETERS:

None

PRE-REQUISITE:

The current crew is set.

DESCRIPTION:

Invoke Crew::isOnShift() API.

RETURN VALUE:

True, when the current crew is on shift.

False, when current crew is not on shift.

DIAGNOSTICS:

Error messages are output to the error log.

Code Mapping Tables

The code mapping tables and views are used to translate values in Oracle Utilities Network Management System to and from the equivalent values in the messages to and from the MDS. These tables are cached to improve performance.

loadMapConfigTable

NAME:

loadMapConfigTable Cache the contents of a table supplying information for the mapping tables.

SYNOPSIS:

loadMapConfigTable(table)

PARAMETERS:

table	A table name.

PRE-REQUISITE:

None.

DESCRIPTION:

Read and cache table.

RETURN VALUE:

True, when successful.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

loadMapTable

NAME:

loadMapTable	Cache the contents of a mapping table.

SYNOPSIS:

loadMapTable(<u>name</u>)

PARAMETERS:

<u>name</u>	The name of the mapping table.

PRE-REQUISITE:

The table, <u>name</u>, exists in the database.

DESCRIPTION:

Read the contents of <u>name</u> and cache its values.

RETURN VALUE:

True, when successful.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

mapTableStr

NAME:

mapTableStr	Return a string given its reference code

SYNOPSIS:

mapTableStr(<u>name</u>, <u>code</u>)

PARAMETERS:

<u>name</u>	A mapping table name
<u>code</u>	A reference code to a string.

PRE-REQUISITE:

The table, <u>name,</u> exists in the database.

DESCRIPTION:

Check if the table, <u>name</u> exists in memory, if not, call loadMapTable to load it.

Look up the string that corresponds to <u>code</u> and return the string. If <u>code</u> is not found, but a default string exists, the default value is returned, otherwise return the empty string.

RETURN VALUE:

The string when successful.

The empty string when code is not found.

DIAGNOSTICS:

Error messages are output to the error log.

mapTableCode

NAME:

mapTableCode   Return a code given its reference string

SYNOPSIS:

mapTableCode(name, string)

PARAMETERS:

name	A mapping table name
string	A string

PRE-REQUISITE:

The table, name, exists in the database.

DESCRIPTION:

Check if the table, name exists in memory, if not, call loadMapTable to load it.

Look up the code that corresponds to string and return the code. If string is not found, but a default code exists, the default code is returned, otherwise return the empty string.

RETURN VALUE:

The code when successful.

The empty string when string is not found.

DIAGNOSTICS:

Error messages are output to the error log.

Damage Assessment Functions

These functions creates and updates damage reports.

findOrCreateDamage

NAME:

findOrCreateDamageCreate a damage report for an device outage

SYNOPSIS:

findOrCreateDamage(event)

PARAMETERS:

event An event handle.

PRE-REQUISITE:

None.

DESCRIPTION:

Load the damage report for event. If no damage report exists for the event, create it.

RETURN VALUE:

True, if the damage report was created.

False, if the damage report existed before the call.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetCrewId

NAME:

damageSetCrewId Update the damage report with a crew ID

SYNOPSIS:

damageSetCrewId(crewId)

PARAMETERS:

crewId A crew ID.

PRE-REQUISITE:

The current damage report is set.

DESCRIPTION:

The crew ID field in the damage report is updated with crewId.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetReportTime

NAME:

damageSetReportTime Update the damage report with a time

SYNOPSIS:

damageSetReportTime(time)

PARAMETERS:

time A time.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The reported time field in the damage report is updated with time.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetAddress

NAME:

damageSetAddress Update the damage report with an Address

SYNOPSIS:

damageSetAddress(Address)

PARAMETERS:

Address An Address.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The Address field in the damage report is updated with Address.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetFeederName

NAME:

damageSetFeederName Update the damage report with a feeder name

SYNOPSIS:

damageSetFeederName(feeder name)

PARAMETERS:

feeder name A feeder name.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The feeder name field in the damage report is updated with feeder name.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetNcg

NAME:

damageSetNcg Update the damage report with an ncg

SYNOPSIS:

damageSetNcg(ncg)

PARAMETERS:

ncg An ncg.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The ncg field in the damage report is updated with ncg.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetTransformer

NAME:

damageSetTransformer   Update the damage report with a transformer

SYNOPSIS:

damageSetTransformer(<u>transformer</u>)

PARAMETERS:

<u>transformer</u>   A grid.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The grid field in the damage report is updated with <u>grid</u>.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetZoneName

NAME:

damageSetZoneName   Update the damage report with a zone name

SYNOPSIS:

damageSetZoneName(<u>zone name</u>)

PARAMETERS:

<u>zone name</u>   A zone name.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The zone name field in the damage report is updated with <u>zone name</u>.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetGrid

NAME:

damageSetGrid Update the damage report with a grid number

SYNOPSIS:

damageSetGrid(grid)

PARAMETERS:

grid A grid number.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The grid field in the damage report is updated with grid.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetSection

NAME:

damageSetSection Update the damage report with the section affected.

SYNOPSIS:

damageSetSection(section)

PARAMETERS:

section The section affected.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The section field in the damage report is updated with section.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetLocation

NAME:

damageSetLocation Update the damage report with the affected location

SYNOPSIS:

damageSetLocation(location)

PARAMETERS:

location	The affected location.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The location field in the damage report is updated with location.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetPhase

NAME:

damageSetPhase Update the damage report with the phases affected

SYNOPSIS:

damageSetLocation(location)

PARAMETERS:

phase	The affected phases.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The phase field in the damage report is updated with phase.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetLoadAffected

NAME:

damageSetLoadAffected   Update the damage report with the affected load.

SYNOPSIS:

damageSetLoadAffected(loadAffected)

PARAMETERS:

loadAffected   The load affected.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The load affected field in the damage report is updated with loadAffected.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetText1

NAME:

damageSetText1   Update a user defined field in the damage report

SYNOPSIS:

damageSetText1(text)

PARAMETERS:

text   A value.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The user defined text #1 field in the damage report is updated with text.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetOption1

NAME:

damageSetOption1 Update a user defined field in the damage report

SYNOPSIS:

damageSetOption1(text)

PARAMETERS:

text A value.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The user defined option #1 field in the damage report is updated with text.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetComment1

NAME:

damageSetComment1 Update the damage report with a comment

SYNOPSIS:

damageSetComment1(text)

PARAMETERS:

text A text.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The comment field in the damage report is updated with text.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetText2

NAME:

damageSetText2 Update a user defined field in the damage report

SYNOPSIS:

damageSetText(text)

PARAMETERS:

text	A value.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The user defined text #2 field in the damage report is updated with text.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

damageSetType

NAME:

damageSetType Update the damage report with the number of affected items

SYNOPSIS:

damageSetType(item, number, accessible)

PARAMETERS:

item	The type of item damaged.
number	Number of item affected.
accessible	Indicate whether the damage is accessible.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

The field containing the number of affected items in the damage report is updated with number and the corresponding accessibility field is updated with accessible

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

saveDamageDetails

NAME:

saveDamage   DetailsSave the damage report

SYNOPSIS:

saveDamageDetails()

PARAMETERS:

None.

PRE-REQUISITE:

The current damage report is set

DESCRIPTION:

Save the current damage report.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

Logging

logLocalError

NAME:

logLocalError   Log an error message to the local log file

SYNOPSIS:

logLocalError(text [, text] …)

PARAMETERS:

text	Text to include in the error message. Any number of parameters can be supplied.

PRE-REQUISITE:

None

DESCRIPTION:

Concatenate all the parameters (no spaces are inserted between the parameters).

Invoke the logError API.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

EXAMPLE:

logLocalError("This is an ", @example, " of an ", &error, " message");

logFatalError

NAME:

logFatalError Log a fatal error message to the local log file and exit

SYNOPSIS:

logFatalError(text [, text] …)

PARAMETERS:

text Text to include in the error message. Any number of parameters can be supplied.

PRE-REQUISITE:

None

DESCRIPTION:

Concatenate all the parameters (no spaces are inserted between the parameters).

Invoke the logFatal API.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

EXAMPLE:

logFatalError("This is an ", @example, " of a fatal ", &error, " message");

logDebug

NAME:

logDebug Log a debug message

SYNOPSIS:

logDebug(level, text)

PARAMETERS:

level The minimum debug level at which to log the message. Zero means always. The debug level of the adapter can be changed by sending it a debug high level message.

text Text to include in the debug message. Any number of parameters can be supplied

PRE-REQUISITE:

None

DESCRIPTION:

Invoke the debug API.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

EXAMPLE:

logDebug(0, "This is an ", @example, " of a ", &debug, " message");

Event Manipulation

These functions read and modify events.

readIncidents

NAME:

readIncidents Populate the Incident Object for the current event

SYNOPSIS:

readIncidents()

PARAMETERS:

None.

PRE-REQUISITE:

The current event is set.

DESCRIPTION:

If the Incident Object for the current event has not been populated previously, and the current event has at least one incident associated with it, invoke the SRS::getIncidents API for the current event.

RETURN VALUE:

The number of incidents in the Incident Object, when successful.

The empty string, when unsuccessful.

API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

clearIncidents

NAME:

clearIncidents Clear the Incident Object, freeing the memory it uses

SYNOPSIS:

clearIncidents()

PARAMETERS:

None.

PRE-REQUISITE:

The current event is set.

DESCRIPTION:

If the Incident Object for the current event has been populated previously, clear it.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

setCaseNoteInfo

NAME:

setCaseNoteInfo Set the case notes for the current order

SYNOPSIS:

setCaseNoteInfo(<u>note</u>)

PARAMETERS:

<u>note</u> Text to be entered in the Case Notes.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

Invoke the SRS::setCaseNoteInfo API for all the events associated with the order.

RETURN VALUE:

True, when successful.

False, when unsuccessful.

API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

getCaseNotesForEvent

NAME:

getCaseNotesForEventGet the case notes for an event

SYNOPSIS:

getCaseNotesForEvent(event)

PARAMETERS:

event	An event handle.

PRE-REQUISITE:

None.

DESCRIPTION:

Invoke the SRS::getCaseNotesForEvent API.

RETURN VALUE:

The case notes for the event when successful.

The empty string when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

setEventInfo

NAME:

setEventInfo	Set event information for an order.

SYNOPSIS:

setEventInfo(outagefield1, value1, [outagefield2, value2], ...)

PARAMETERS:

outagefield1	The outage field to update/set.
value1	The value to set.
outagefield2	The outage field to update/set.
value2	The value to set.

PRE-REQUISITE:

The database table 'OUTAGE_FIELD' must be defined and populated. It contains the valid outage fields that can be used in outagefield.

DESCRIPTION:

Update outagefield[1,2…] with value[1,2…] for event. Multiple outagefields and values updates are supported. The SRS::setFieldInfo(..) API is invoked for all events associated with the order.

RETURN VALUE:

True, when successful.

False when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

completeEvent

NAME:

completeEvent Complete all events associated with the current order

SYNOPSIS:

completeEvent()

PARAMETERS:

None.

PRE-REQUISITE:

The current order is set.DESCRIPTION:

Restore and complete the event(s) associated with the current order.

Because the state of the event changes when the API's used by this function are called, the Event Object is automatically reloaded after the call. This means that the Incident Object will not be populated. If the Incident Object is required, it must be populated using readIncident().

RETURN VALUE:

True, when successful.

False, when unsuccessful.

Could not restore event.

DIAGNOSTICS:

Error messages are output to the error log.

setGenericField

NAME:

setGenericField Update event information for the current order

SYNOPSIS:

setGenericField(field, value, user)

PARAMETERS:

field	A field to update.
value	A value.
user	Who initiated the update.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

For all events associated with the current order, update field with value, indicating that user initiated the update.

The SRS API 'setGenericField()' is invoked.

RETURN VALUE:

True, when successful.

False, when the API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

readGenericField

NAME:

readGenericField Read information for the current event

SYNOPSIS:

readGenericField(<u>field</u>)

PARAMETERS:

<u>field</u> A field to read.PRE-REQUISITE:

The current event is set. The value of <u>field</u> is a valid generic field name.

DESCRIPTION:

Read the value of <u>field</u> for the current event.

The SRSoutput API 'getGenericField()' is invoked.

RETURN VALUE:

The value of <u>field</u>.

DIAGNOSTICS:

Error messages are output to the error log.

ert

NAME:

ert Set the estimated restoration time for the current order

SYNOPSIS:

ert(<u>time</u>)

PARAMETERS:

<u>time</u> The estimated restoration time in internal format.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

Call the SRS:: setEstRestTime() API for all events associated with the current order.

RETURN VALUE:

True, when successful.

False, when the API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

requestRowAction

NAME:

requestRowAction Press a button for the current event

SYNOPSIS:

requestRowAction(table, button)

PARAMETERS:

table	The name of the table (work_agenda is most common).
button	The name of the button to press.

PRE-REQUISITE:

The current event is set.

DESCRIPTION:

Call the TES::requestRowAction API for the current event.

RETURN VALUE:

True, when the API call succeeds.

False, when the API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

requestRowActionAll

NAME:

requestRowActionAll Press a button for the current order

SYNOPSIS:

requestRowActionAll(table, button)

PARAMETERS:

table	The name of the table (work_agenda is most common).
button	The name of the button to press.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

Call the TES::requestRowAction API for all events associated with the current order.

RETURN VALUE:

True, when the API call succeeds.

False, when the API call fails.

DIAGNOSTICS:

Error messages are output to the error log.

eventIsActive

NAME:

eventIsActive   Check that the current order has at least one active event

SYNOPSIS:

eventIsActive()

PARAMETERS:

None.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

Determine if the order has at least one active event.

RETURN VALUE:

True, when active.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

confirmDeviceOutage

NAME:

confirmDeviceOutage   Confirm a device outage for all events associated with the current order.

SYNOPSIS:

confirmDeviceOutage(<u>phases</u>)

PARAMETERS:

<u>phases</u>	The phases that are out

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

For all events associated with the current order, confirm that it is a real device outage. If <u>phases</u> are all the device's phases, call the SRS::convertPOToRO() API, otherwise open the phases on the device, using the DDS::operateState() API.

Because the state of the event changes when these API's are called, the Event Object is automatically reloaded after the call. This means that the Incident Object will not be populated. If the Incident Object is required, it must be populated using readIncident().

RETURN VALUE:

True, when confirmation is successful.

False, if an API call fails. This will occur if the device has tags which prevent opening the device.

DIAGNOSTICS:

Error messages are output to the error log.

confirmServiceOutage

NAME:

confirmServiceOutage   Confirm a service outage for all events associated with the current order.

SYNOPSIS:

confirmServiceOutage()

PARAMETERS:

None.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

For all events associated with the current order, confirm that the customers described in the incidents read for the event, are individually out using the SRS::updateOutageTimes() API. Note that no outages are created for customers who are not attached to the device (for example a fuzzy call).

Because the state of the event changes when this API is called, the Event Object is automatically reloaded after the call. This means that the Incident Object will not be populated. If the Incident Object is required, it must be populated using readIncident().

Because this function can create events, lockForEventCreation() should be called if the new events should not have orders created for them. If a pseudo relationship is to be created from the resulting events, it is recommended that createPseudoRelationFromConfirmServiceOutage() as it does not require the use of lockForEventCreation().

SEE ALSO:

createPseudoRelationFromConfirmServiceOutage(), section **createPseudoRelationFromConfirmServiceOutage** on page 7-106, and lockForEventCreation() section **lockForEventCreation** on page 7-103.

RETURN VALUE:

The number of customers confirmed, when confirmation is successful.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

lockForEventCreation

NAME:

lockForEventCreation   Prevent the processing of new events until the current document is fully processed.

SYNOPSIS:

lockForEventCreation()

PARAMETERS:

None.

PRE-REQUISITE:

None.

DESCRIPTION:

Prevent new events from being processed until the current document finishes processing. This is required if a call can cause events, e.g., confirmServiceOutage(), and the new events need to be processed, e.g., by createPseudoRelation(). If possible avoid using this function, because it prevents other threads from processing any changes to events.

SEE ALSO:

confirmServiceOutage(), section **confirmServiceOutage** on page 7-102, and createPseudoRelation(), section **createPseudoRelation** on page 7-106.

RETURN VALUE:

The empty string.

False, otherwise.

DIAGNOSTICS:

None.

restoreOutage

NAME:

restoreOutage   Restore all events for the current order

SYNOPSIS:

restoreOutage()

PARAMETERS:

None.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

For all events associated with the current order restore the event. If the event is a device outage restore it by closing all of the device's phases using the DDS::operateState() API. If the event is a service outage, restore it using the SRS::processIndivServUpdate() API.

Because the state of the event changes when these API's are called, the Event Object is automatically reloaded after the call. This means that the Incident Object will not be populated. If the Incident Object is required, it must be populated using readIncident().

RETURN VALUE:

True, when restoration is successful.

False, if an API fails. This will occur in a device outage if the device has tags which prevent closing the device. This may occur in a service outage if the event has not been acknowledged.

DIAGNOSTICS:

Error messages are output to the error log.

picklistCompLog

NAME:

picklistCompLog Update the database table picklist_completion_log for all the events associated with the current order.

SYNOPSIS:

picklistCompLog(who, reason)

PARAMETERS:

who	Who performed the action.
reason	What occurred

PRE-REQUISITE:

Current order is set.

DESCRIPTION:

For all events associated with the current order, create an entry in the database table 'picklist_completion_log' containing who and reason.

RETURN VALUE:

True, when successful.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

picklistInfoUpdTr

NAME:

picklistInfoUpdTr Update the database table picklist_info_upd_tr for the current order.

SYNOPSIS:

picklistInfoUpdTr(field1, value1, [field2, value2], ..)

PARAMETERS:

field[1,2,..]	Fields to update
value[1,2,..]	Assignment Values

PRE-REQUISITE:

Current order is set.

DESCRIPTION:

For all events associated with the current order, update the field s with value s in the database table 'picklist_info_upd_tr'.

RETURN VALUE:

True, when successful.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

Relation Functions

These functions manipulate dml and aggregate relationships.

Where these functions take a type parameter, it must be one of: NESTED_OUTAGE, MOMENTARY_OUTAGE, PARTIAL_RESTORATION, and RELATED_OUTAGE. (PSEUDO_ NESTED_OUTAGE, MOMENTARY_OUTAGE, PARTIAL_RESTORATION, and RELATED_OUTAGE are valid, but have the same effect as their non-pseudo counterparts).

findRelation

NAME:

findRelation	Find a dml relation by matching the contents of an relation object field. If found, set the current relation object to the relation found.

SYNOPSIS:

findRelation(type, fieldname, value)

PARAMETERS:

type	A relationship type.
fieldname	A field name
value	A value

PRE-REQUISITE:

The type parameter is valid.

If the current order is set, the current order must be in the relation. (This can be guaranteed by not calling findOrder, or by finding the relation by order's key event using findRelation(event, $O.event).)

DESCRIPTION:

Find the relation with a type of type whose fieldname has a value of value.

RETURN VALUE:

True, when successful.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

createPseudoRelation

NAME:

createPseudoRelation   Create a pseudo (non-Oracle Utilities Network Management System) relationship.

SYNOPSIS:

createPseudoRelation(type)

PARAMETERS:

type	A relationship type.

PRE-REQUISITE:

The current order is set

The type parameter is valid.

The relationship type must have been configured for aggregate processing (this may be changed in a future release).

DESCRIPTION:

Create a pseudo relationship of type among all outage events whose device is that of the current order's active event.

RETURN VALUE:

True, when successful.

False, when unsuccessful.

No events exist on device.

The order's event is already in another relation.

All the other events on the device are in another relation.

DIAGNOSTICS:

Error messages are output to the error log.

createPseudoRelationFromConfirmServiceOutage

NAME:

createPseudoRelationFromConfirmServiceOutage Create a pseudo (non-Oracle Utilities Network Management System) relationship from the results of the confirmation of a service outage.

SYNOPSIS:

createPseudoRelationFromConfirmServiceOutage(type)

PARAMETERS:

type	A relationship type.

PRE-REQUISITE:

See confirmServiceOutage, section **confirmServiceOutage** on page 7-102 and createPseudoRelation, section **createPseudoRelation** on page 7-106.

DESCRIPTION:

Confirm a service outage as described in confirmServiceOutage, section **confirmServiceOutage** on page 7-102.

Create a pseudo relationship of type as described in createPseudoRelation, section **createPseudoRelation** on page 7-106.

This combined function is recommended rather than calling confirmServiceOutage and then calling createPseudoRelation because:

There is no need to call lockForEventCreation()

If this call is in progress when the adapter exits, it will be completed fully when the adapter restarts.

RETURN VALUE:

The number of customers confirmed, when successful.

False, when unsuccessful.

No events exist on device.

The order's event is already in another relation.

All the other events on the device are in another relation.

DIAGNOSTICS:

Error messages are output to the error log.

triggerRelationChanged

NAME:

triggerRelationChanged   Trigger all output documents with a RelationChanged trigger.

SYNOPSIS:

triggerRelationChanged(relation)

PARAMETERS:

relation	The relation's handle. (If the relation has been found, $R.relation gives this value).

PRE-REQUISITE:

There is at least one output document with a RelationChanged trigger.

DESCRIPTION:

For all events in the relation, trigger all output documents with a RelationChanged trigger, with the event's handle as the trigger argument.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

deleteRelation

NAME:

deleteRelation Delete the current relation.

SYNOPSIS:

deleteRelation()

PARAMETERS:

None.

PRE-REQUISITE:

The current relation is set.

DESCRIPTION:

Delete the current relation. After the deletion, there is no current relation.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

Miscellaneous API Functions

classTable

NAME:

classTable Return the class table for a class.

SYNOPSIS:

classTable(class)

PARAMETERS:

class The class number

PRE-REQUISITE:

The class parameter must be an integer.

DESCRIPTION:

Call the ODS::getTable() API. If the table does not exist an empty string is returned.

RETURN VALUE:

The table name, when successful.

The empty string when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

getClassDesc

NAME:

getClassDesc Returns a textual description of the Class.

SYNOPSIS:

getClasDesc(class)

PARAMETERS:

class The class number

PRE-REQUISITE:

The class parameter must be an integer.

DESCRIPTION:

Call the ODS::getClassDesc() API. If the class does not exist an empty string is returned.

RETURN VALUE:

The textual description, when successful.

The empty string when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log

isCls

NAME:

isCls Check if a class is one of classes in a list.

SYNOPSIS:

isCls(class, className1, className2, …)

PARAMETERS:

class A class number.

className1, className2, …A list of class names.

PRE-REQUISITE:

None

DESCRIPTION:

Read and cache the class numbers for all the className parameters, using the ODS::getClassIndex API.

If class is one of the class numbers, return true, false otherwise.

RETURN VALUE:

True, when class is in the list.

False, when class is not in the list.

DIAGNOSTICS:

Error messages are output to the error log.

setAlarm

NAME:

setAlarm   Send an alarm to the WorkAgenda

SYNOPSIS:

setAlarm(deviceHandle, alarmMsg)

PARAMETERS:

deviceHandle   A device handle.

alarmMsg   A alarm message.

PRE-REQUISITE:

None

DESCRIPTION:

Send an alarmMsg regarding deviceHandle, using the DDS::sendAlarm API.

RETURN VALUE:

None.

DIAGNOSTICS:

Error messages are output to the error log.

getGuid

NAME:

getGuid   Return a globally unique id.

SYNOPSIS:

getGuid()

PARAMETERS:

None

PRE-REQUISITE:

None

DESCRIPTION:

Invokes GatewayUtil::CreateGuid() API.

RETURN VALUE:

The GUID, when successful.

The empty string when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

interfaceUp

NAME:

interfaceUp Register the state of the interface

SYNOPSIS:

interfaceUp()

PARAMETERS:

None.

PRE-REQUISITE:

None

DESCRIPTION:

Register that the interface is currently up. Invoke SMS::registerCallback(), and SMS::registerInterfaceFailed() API.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

interfaceDown

NAME:

interfaceDown Register the state of the interface

SYNOPSIS:

interfaceDown()

PARAMETERS:

None.

PRE-REQUISITE:

None

DESCRIPTION:

Register that the interface is currently down. Invoke SMS::registerCallback(), and SMS::registerInterfaceFailed() API.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

sql

NAME:

sql Execute a non-select SQL statement

SYNOPSIS:

sql(sqlStatement)

PARAMETERS:

sqlStatement A non-select SQL statement.

PRE-REQUISITE:

None

DESCRIPTION:

Execute sqlStatement using the DBS::sql() API.

RETURN VALUE:

True, when successful.

False, when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

EXAMPLE:

```
sql("delete from damage_report where event_cls=" + $E.outageHdl.cls +
      " and event_idx=" + $E.outageHdl.idx);
```

Non API Functions

This set of functions does not use the Oracle Utilities Network Management System API.

isSet

NAME:

isSet Check if a parameter has been set.

SYNOPSIS:

isSet(param)

PARAMETERS:

param The parameter to check.

PRE-REQUISITE:

None

DESCRIPTION:

Check if the param has been set.

RETURN VALUE:

True, if param has been set.

False, if not.

DIAGNOSTICS:

None.

length

NAME:

length	Return the number of characters in a string.

SYNOPSIS:

length(string)

PARAMETERS:

string	A string value

PRE-REQUISITE:

None

DESCRIPTION:

Determine the length of string.

RETURN VALUE:

The length of string.

DIAGNOSTICS:

None.

substring

NAME:

substring	Return a sub-string of a value

SYNOPSIS:

substring(string, start, length)

PARAMETERS:

string	A string value
start	The starting position of the subset in string
length	The length of the sub-string to return

PRE-REQUISITE:

None

DESCRIPTION:

Return a subset of string whose size equals length, and starts at position start in string.

If length is less than 1, return the empty string.

If start is zero or positive, it is an offset from the start of string.

If start is negative, it is an offset from the end of string.

If there are less than length characters in string starting at start, return all the characters in string starting at start.

Otherwise return length characters from string starting at start.

RETURN VALUE:

The sub-string

DIAGNOSTICS:

None.

stringbefore

NAME:

stringbefore Return a sub-string of a string value

SYNOPSIS:

stringbefore(string, stop)

PARAMETERS:

string	A string value
stop	A string to stop at.

PRE-REQUISITE:

None

DESCRIPTION:

Search string for stop and return all characters before stop. If stop does not exist within string return string

RETURN VALUE:

The sub-string.

DIAGNOSTICS:

None.

isDigits

NAME:

isDigits Check if the string is made up of digits only

SYNOPSIS:

isDigits(string)

PARAMETERS:

string	A string value

PRE-REQUISITE:

None

DESCRIPTION:

Check if string is made up of purely numeric values (i.e. 0 to 9)

RETURN VALUE:

True, if string is all digits.

False, otherwise.

DIAGNOSTICS:

Error messages are output to the error log.

stringInString

NAME:

stringInString Check to see whether a sub-string exists in another string

SYNOPSIS:

stringInString(string1, string2)

PARAMETERS:

string1	A string value
string2	A string value

PRE-REQUISITE:

None

DESCRIPTION:

Search string2 for string1.

RETURN VALUE:

True, if string1 is found

False, otherwise.

DIAGNOSTICS:

None.

removeDelim

NAME:

removeDelim Return a substring without the contents contained within the delimiters, including the delimiters.

SYNOPSIS:

removeDelim(string, start, end)

PARAMETERS:

string	A string value
start	A starting delimiter
end	A end delimiter

PRE-REQUISITE:

None

DESCRIPTION:

Search string for start, remove all characters found between and including start and end. If start is not found return string. If end is not found, return all characters after and including start.

RETURN VALUE:

The sub-string value.

DIAGNOSTICS:

Error messages are output to the error log.

decodeDateTime

NAME:

decodeDateTime Translate a formatted time string into internal format.

SYNOPSIS:

decodeDateTime(time)

PARAMETERS:

time	A time in the format *yyyy-mm-dd*T*hh*:*mm*:*ss*

PRE-REQUISITE:

None

DESCRIPTION:

Return time in internal time format. If time is not in the correct format, return the empty string.

RETURN VALUE:

The time in internal format, when successful.

False. when unsuccessful.

Format of time is invalid

DIAGNOSTICS:

Error messages are output to the error log.

formatDateTime

NAME:

formatDateTime Translate a time in internal format into a formatted time string.

SYNOPSIS:

formatDateTime(time)

PARAMETERS:

time	A time in internal format.

PRE-REQUISITE:

None

DESCRIPTION:

Format time to *yyyy-mm-dd*T*hh*:*mm*:ss. If time is not in internal format, return the empty string.

RETURN VALUE:

The formatted time, when successful.

False. when unsuccessful

Invalid time supplied.

DIAGNOSTICS:

Error messages are output to the error log.

formatDTNow

NAME:

formatDTNow   Format the current system time

SYNOPSIS:

formatDTNow()

PARAMETERS:

None.

PRE-REQUISITE:

None

DESCRIPTION:

Return the results of formatDateTime(time()).

RETURN VALUE:

The formatted time.

DIAGNOSTICS:

None.

time

NAME:

time   Return the current system time

SYNOPSIS:

time()

PARAMETERS:

None

PRE-REQUISITE:

None

DESCRIPTION:

Return the current system time

RETURN VALUE:

The current time, in internal format.

DIAGNOSTICS:

None.

pause

NAME:

pause Pause evaluation for a period of time

SYNOPSIS:

pause(seconds)

PARAMETERS:

seconds The number of seconds to pause

PRE-REQUISITE:

None

DESCRIPTION:

Pause for seconds.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

isIn

NAME:

isIn Check if a value exists in a list

SYNOPSIS:

isIn(value, item1, item2, item3, …)

PARAMETERS:

value A value

item1, item2, item3, …A list of values.

PRE-REQUISITE:

None

DESCRIPTION:

Check item1, item2, item3, … for value.

RETURN VALUE:

True, when value is found.

False, when not found.

DIAGNOSTICS:

None.

selectValue

NAME:

selectValue Select a value based on a input string.

SYNOPSIS:

selectValue(string, default, match1, value1, [match2, value2], [match3, value3], …)

PARAMETERS:

string	A value to match
default	A default value
match[123]	A list of values to compare to
value[123]	The values to return if a match is found

PRE-REQUISITE:

None

DESCRIPTION:

Search match for string and return the corresponding value. If string is not found return default.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

NOTES:

For large lists, the use of a code mapping table would be more appropriate.

EXAMPLE:

```
@exp = selectValue(3, None, 1, “hello”, 2, “goodbye”, 3, “later”)
```

Therefore, string = 3,default = None,match = 1, 2, 3 value = hello, goodbye, later

In this example, @exp = later.

triggerOutputDoc

NAME:

triggerOutputDoc Trigger an output document

SYNOPSIS:

triggerOutputDoc(doc, trig, [argument1, argument2, …])

PARAMETERS:

doc	An output document
trig	The name of the trigger to fire.
argument	Argument required to triggered the output document.

PRE-REQUISITE:

A document named doc must exist.

There must be an OnRequest trigger named trig in doc.

The number of arguments must match the number of arguments expected by trig.DESCRIPTION:

Validate doc and argument. Queue doc for processing.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

sortIncidents

NAME:

sortIncidents Sort the incidents in the current event

SYNOPSIS:

sortIncidents(field1, sort order1, [field2, sort order2, …])

PARAMETERS:

field	An incident object field
sort order	The order to sort in, 'asc' meaning ascending or 'desc' meaning descending (this parameter is not case sensitive).

PRE-REQUISITE:

The current event is set.

The fields parameters must be valid incident fields See **Incident Object Fields** on page 7-133, below for the available incident fields.

The order parameters must be valid. If the last order parameter is omitted it defaults to 'asc'

DESCRIPTION:

Sort the incidents in the current event. When two incidents are compared, the specified fields are compared in the order they appear in the parameter list. If they differ, the incident with the lower value comes first in the list if it has an ascending order, otherwise the incident with the higher value comes first. If the two fields are equal, the sort order depends on the next field in the parameter list. If all fields are equal, the order of the two incidents is undetermined. To force a consistent order, it is recommended that the last field be the 'getCondHdl' field which always differ (the newer incident having the larger value).

All comparisons are based on the internal types of the fields, in order to give the expected results. Character data is sorted in lexical order, the case being significant.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

highPriTCCategoriesFromClues

NAME:

highPriTCCategoriesFromClues Return the highest priority trouble code categories from all of the supplied clues.

SYNOPSIS:

highPriTCCategoriesFromClues(<u>clues</u>)

PARAMETERS:

<u>clues</u>	The clues to decode

PRE-REQUISITE:

The MDS_HIGH_PRI_CAT table must exist in the database.

DESCRIPTION:

If the MDS_HIGH_PRI_CAT table has not been read, read and cache its contents. This table supplies the priority order for each group in the trouble code.

Decode each clue into it's group and numeric value

Find the highest priority for each group in all the clues, and assemble these into a composite trouble code.

RETURN VALUE:

The composite trouble code.

DIAGNOSTICS:

Error messages are output to the error log.

loadTroubleCodes

NAME:

loadTroubleCodes Cache the trouble codes and their equivalent textual descriptions.

SYNOPSIS:

loadTroubleCodes()

PARAMETERS:

None.

PRE-REQUISITE:

The table SRS_TROUBLE_CODES exists in the database.

DESCRIPTION:

The trouble codes are cached in-groups using the 'group_order' column. For each group the 'code_num' column and the 'short_desc' column are cached. The 'code_num' is used as the trouble codes' reference code and the 'short_desc' is used as the trouble codes' textual description.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

formatClues

NAME:

formatClues   Convert a trouble code into a textual description

SYNOPSIS:

formatClues(<u>trCode</u>)

PARAMETERS:

<u>trCode</u>   A trouble code

PRE-REQUISITE:

The table SRS_TROUBLE_CODES exists in the database.

DESCRIPTION:

If the SRS_TROUBLE_CODES table has not been read, read and cache its contents by calling loadTroubleCodes().

Convert each digit in <u>trCode</u> to its equivalent textual description. Concatenate the descriptions.

RETURN VALUE:

The textual description.

DIAGNOSTICS:

None.

phaseStr

NAME:

phaseStr   Convert a set of phases in internal bitmap format to a textual representation.

SYNOPSIS:

phaseStr(<u>phases</u>)

PARAMETERS:

<u>phases</u>   The phases in internal bitmap format.

PRE-REQUISITE:

None

DESCRIPTION:

Convert the phase bits into 'ABC' format. If the bit for a phase is not set, do not include its letter.

RETURN VALUE:

The textual representation.

DIAGNOSTICS:

None.

phaseInt

NAME:

phaseIntConvert a set of phases in a textual representation to an internal bitmap format.

SYNOPSIS:

phaseInt(phases)

PARAMETERS:

phases	A textual representation of the phases.

PRE-REQUISITE:

None

DESCRIPTION:

Convert the textual phase in 'ABC' format to its internal bitmap format.

RETURN VALUE:

The internal bitmap format.

DIAGNOSTICS:

None.

setTimeout

NAME:

setTimeout	Set a time out to call a function

SYNOPSIS:

setTimeout(name, wait, function)

PARAMETERS:

name	The name of the time out.
wait	The time to wait, in seconds.
function	The function to call at when the time expires.

PRE-REQUISITE:

The wait parameter must be an integer greater than zero.

The function parameter must be a function call.

DESCRIPTION:

If there is an un-expired timeout with the same name, do nothing.

Evaluate all the parameters of the function, if any.

Start a timeout with the specified name.

Call the function when the timeout expires (unless cancelled by cancelTimeout()).

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

cancelTimeout

NAME:

cancelTimeout Cancel a timeout.

SYNOPSIS:

cancelTimeout(name)

PARAMETERS:

name The name of the timeout to cancel.

PRE-REQUISITE:

None

DESCRIPTION:

If there is a timeout with the specified name, cancel it, preventing the timeout's function from being called, otherwise, do nothing.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

createOrder

NAME:

createOrder Create an order.

SYNOPSIS:

createOrder()

PARAMETERS:

None.

PRE-REQUISITE:

The current event is set(usually due to the arrival of an SRSoutput message).

The current event must not be associated with another order. This can be determined by a call to findOrder(event, $E.outageHdl).

If the current relation is set, the current event must be in the relation. (This can be guaranteed by not calling findRelation, or by finding the relation by relation's key event using findRelation(event, $E.outageHdl.)

DESCRIPTION:

Create the order's row in the MDS_ORDER table, and associate the event with the order.

Populate internal data structures for the order.

Set the current order to the order created.

RETURN VALUE:

True if successful.

False if unsuccessful (current event already associated with another order).

DIAGNOSTICS:

Error messages are output to the error log.

findOrder

NAME:

findOrder	Find an order by matching the contents of an order object field. If found, set the current order object to the order found.

SYNOPSIS:

findOrder(field, value)

PARAMETERS:

field	An order object field name.
value	The value to match

PRE-REQUISITE:

The field parameter must be a valid order field name.

If the current relation is set, the order must be in the relation. (This can be guaranteed by not calling findRelation, or by finding the order from the relation's key event, using findOrder(event, $R.event).)

DESCRIPTION:

Search for an order with a field whose value is value.

If found, set the current order object to the order that was found and return true.

If none is found, return false.

haveOrder

NAME:

haveOrder	Determine if there is an order matching the contents of an order object field, without entering any mutexes.

SYNOPSIS:

haveOrder(field, value)

PARAMETERS:

field	An order object field name.
value	The value to match

PRE-REQUISITE:

The field parameter must be a valid order field name.

DESCRIPTION:

Search for an order with a field whose value is value.

RETURN VALUE:

True, when successful.

False, when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

completeOrder

NAME:

completeOrder Complete the current order, making it no longer active.

SYNOPSIS:

completeOrder(text)

PARAMETERS:

text	A description of the why the order is complete. For example, the order could have been cancelled or completed by the crew.

PRE-REQUISITE:

The current order is set.

DESCRIPTION:

Complete the order by setting its ‘active’ column to ‘N’, its ‘when_completed’ column to the current time, and setting its ‘comp_reason’ column to text in MDS_ORDER table. Clear all data structures relating to the order.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

haveEventOrder

NAME:

haveEventOrder Determine whether these is an order for an event.

SYNOPSIS:

haveEventOrder(event)

PARAMETERS:

event   The event's handle.

PRE-REQUISITE:

None.

DESCRIPTION:

Determine whether there is an order for the event handle is event.

RETURN VALUE:

True, if the event has an order.

False, if the event does not have an order.

DIAGNOSTICS:

None.

findEventObject

NAME:

findEventObject   Find the External Event Object for an event.

SYNOPSIS:

findEventObject(event)

PARAMETERS:

event   The event's handle.

PRE-REQUISITE:

None.

DESCRIPTION:

Find the event object whose handle is event. If successful, make it the current event.

RETURN VALUE:

True, when successful.

False, when unsuccessful.

DIAGNOSTICS:

Error messages are output to the error log.

setDocValue

NAME:

setDocValue   Change the value of an element, attribute or variable in an active document

SYNOPSIS:

setDocValue(object, doc, name, value)

PARAMETERS:

object	The object identifier character for the object that holds the active document. Currently only ‘O’, the order object, is available.
doc	The name of the document
name	The alternate name of the entity whose value is to be set.
value	The value to set the entity to.

PRE-REQUISITE:

object must be a valid object identifier, and it must be a current object (for example, use findOrder()).

doc must be a associated with the object.

name must be the alternate name of an entity in doc.

DESCRIPTION:

Find the active document doc in the current object.

In the document set the entity named name to value.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

Error messages are output to the error log.

printEvntCrew

NAME:

printEvntCrew   Print the current assignments/dispatches of crews to orders to the log.

SYNOPSIS:

printEvntCrew()

PARAMETERS:

None.

PRE-REQUISITE:

None.

DESCRIPTION:

Print the crews assigned and dispatched to all orders to the log.

RETURN VALUE:

The empty string.

DIAGNOSTICS:

None.

xml

NAME:

xml Return the current input xml document, if any.

SYNOPSIS:

xml()

PARAMETERS:

None.

PRE-REQUISITE:

None.

DESCRIPTION:

Return the current input document, or the empty string, if none.

RETURN VALUE:

The current input xml document, if any.

The empty string, when an input xml document in not being processed.

DIAGNOSTICS:

None.

Event Object Fields

These are the fields available in the external event object ‘E’. For each field the equivalent SRSoutput data field is listed.

Field Name	SRSoutput Data Fields
alarmHdl	SRSoutput::alarmHdl
devHdl	SRSoutput::devHdl
feederHdl	SRSoutput::feederHdl
outageHdl	SRSoutput::outageHdl
oldEvent	SRSoutput::oldEvent
condHdl	SRSoutput::condHdl
extraDevHdl	SRSoutput::extraDevHdl
devClsName	SRSoutput::devClsName
incidentType	SRSoutput::incidentType
description	SRSoutput::description
cause	SRSoutput::cause
preferredAlias	SRSoutput::perferredAlias

Field Name	SRSoutput Data Fields
troubleCode	SRSoutput::troubleCode
troubleQueue	SRSoutput::troubleQueue
office	SRSoutput::office
circuit	SRSoutput::circuit
district	SRSoutput::district
dispAddress	SRSoutput::dispAddress
addrBuilding	SRSoutput::addrBuilding
addrStreet	SRSoutput::addrStreet
addrCity	SRSoutput::addrCity
city	SRSoutput::city
status	SRSoutput::status
feeder	SRSoutput::feeder
tags	SRSoutput::tags
est_source	SRSoutput::est_source
comment	SRSoutput::comment
devCode	SRSoutput::devCode
externId	SRSoutput::externId
crewId	SRSoutput::crewId
supplyNodeDevice	SRSoutput::supplyNodeDevice
supplyNodeIndexes	SRSoutput::supplyNodeIndexes
leafNcgs	SRSoutput::leafNcgs
numLeafNcgs	SRSoutput::numLeafNcgs
actionIDs	SRSoutput::actionIDs
numActionsIDs	SRSoutput::numActionsIDs
outageTime	SRSoutput::outageTime
firstIncTime	SRSoutput::firstIncTime
firstIncTimeStr	SRSoutput::firstIncTimeStr
estRestTime	SRSoutput::estRestTime
estRestTimeStr	SRSoutput::estRestTimeStr
estAssessTime	SRSoutput::estAssessTime
complete_time	SRSoutput::complete_time
job_complete_time	SRSoutput::job_complete_time
msgType	SRSoutput::msgType

Field Name	SRSoutput Data Fields
troubleCode	SRSoutput::troubleCode
troubleQueue	SRSoutput::troubleQueue
office	SRSoutput::office
circuit	SRSoutput::circuit
district	SRSoutput::district
dispAddress	SRSoutput::dispAddress
addrBuilding	SRSoutput::addrBuilding
addrStreet	SRSoutput::addrStreet
addrCity	SRSoutput::addrCity
city	SRSoutput::city
status	SRSoutput::status
feeder	SRSoutput::feeder
tags	SRSoutput::tags
est_source	SRSoutput::est_source
comment	SRSoutput::comment
devCode	SRSoutput::devCode
externId	SRSoutput::externId
crewId	SRSoutput::crewId
supplyNodeDevice	SRSoutput::supplyNodeDevice
supplyNodeIndexes	SRSoutput::supplyNodeIndexes
leafNcgs	SRSoutput::leafNcgs
numLeafNcgs	SRSoutput::numLeafNcgs
actionIDs	SRSoutput::actionIDs
numActionsIDs	SRSoutput::numActionsIDs
outageTime	SRSoutput::outageTime
firstIncTime	SRSoutput::firstIncTime
firstIncTimeStr	SRSoutput::firstIncTimeStr
estRestTime	SRSoutput::estRestTime
estRestTimeStr	SRSoutput::estRestTimeStr
estAssessTime	SRSoutput::estAssessTime
complete_time	SRSoutput::complete_time
job_complete_time	SRSoutput::job_complete_time
msgType	SRSoutput::msgType

Field Name	SRSoutput Data Fields
validStateKey	SRSoutput::validStateKey
prevStateKey	SRSoutput::prevStateKey
stateValue	SRSoutput::stateValue
condStatus	SRSoutput::condStatus
condPhases	SRSoutput::condPhases
customersOut	SRSoutput::customersOut
typeMask	SRSoutput::typeMask
partition	SRSoutput::partition
ncg	SRSoutput::ncg
appliedRule	SRSoutput::appliedRule
priority	SRSoutput::priority
custCall	SRSoutput::custCall
pri_w	SRSoutput::pri_w
pri_sw	SRSoutput::pri_sw
pri_p	SRSoutput::pri_p
pri_e	SRSoutput::pri_e
custCrit	SRSoutput::custCrit
crit_1	SRSoutput::crit_1
crit_2	SRSoutput::crit_2
crit_3	SRSoutput::crit_3
crit_tot	SRSoutput::crit_tot
revenue	SRSoutput::revenue
numSndDev	SRSoutput:: numSndDev
numSupplied	SRSoutput::numSupplied
outageRef	SRSoutput::outageRef
oldCondID	SRSoutput::oldCondID
sort_col_1	SRSoutput::sort_col_1
sort_col_2	SRSoutput::sort_col_2
sort_col_3	SRSoutput::sort_col_3
sort_col_4	SRSoutput::sort_col_4
sort_col_5	SRSoutput::sort_col_5
sort_col_6	SRSoutput::sort_col_6
sort_col_7	SRSoutput::sort_col_7

Field Name	SRSoutput Data Fields
sort_col_8	SRSoutput::sort_col_8
sort_col_9	SRSoutput::sort_col_9
life_support	SRSoutput::life_support
outage_type	SRSoutput::outage_type
old_outage_type	SRSoutput::old_outage_type
group_type	SRSoutput::group_type
messages	SRSoutput::messages
customer_phone	SRSoutput::customer_phone
numb	SRSoutput::numb
inc_outage	SRSoutput::inc_outage
devAlias	SRSoutput::devAlias
customerName	SRSoutput::customerName
rule_set	SRSoutput::rule_set
crit_k	SRSoutput::crit_k
crit_c	SRSoutput::crit_c
crit_d	SRSoutput::crit_d
from_str	SRSoutput::from_str
mergedEvents	SRSoutput::mergedEvent
numMerged	SRSoutput::numMerged
hasClue	SRSoutput::hasClue
ctrlZoneHdl1	SRSoutput::ctrlZoneHdl1
ctrlZoneHdl2	SRSoutput::ctrlZoneHdl2
ctrlZoneHdl3	SRSoutput::ctrlZoneHdl3
ctrlZoneHdl4	SRSoutput::ctrlZoneHdl4
ctrlZoneHdl5	SRSoutput::ctrlZoneHdl5
ctrlZoneHdl6	SRSoutput::ctrlZoneHdl6
ctrlZoneName1	SRSoutput::ctrlZoneName1
ctrlZoneName2	SRSoutput::ctrlZoneName2
ctrlZoneName3	SRSoutput::ctrlZoneName3
ctrlZoneName4	SRSoutput::ctrlZoneName4
ctrlZoneName5	SRSoutput::ctrlZoneName5
ctrlZoneName6	SRSoutput::ctrlZoneName6
who_responsible	SRSoutput::who_responsible

Field Name	SRSoutput Data Fields
who_completed	SRSoutput::who_completed
resp_modify_time	SRSoutput:: resp_modify_time
xRef	SRSoutput::xRef
yRef	SRSoutput::yRef
highestIncPri	SRSoutput::highestIncPri
referralGroup	SRSoutput::referralGroup
who	SRSoutput::who
firstCrewTime	SRSoutput::firstCrewTime
additionalDevHdls	SRSoutput::additionalDevHdls
additionalCondHdls	SRSoutput::additionalCondHdls
relatedEvents	SRSoutput::relatedEvents
additionalCondStatuses	SRSoutput::additionalCondStatuses
numAdditional	SRSoutput::numAdditional
numRelated	SRSoutput::numRelated
maxNum	SRSoutput::maxNum
maxNumRelated	SRSoutput::maxNumRelated
routeId	SRSoutput::routeId
repair_minutes	SRSoutput::repair_minutes
crew_eta	SRSoutput::crew_eta
sheetNums	SRSoutput::sheetNums
generic_col_1	SRSoutput::generic_col_1
related_event	SRSoutput::related_event
related_type	SRSoutput::related_type

Incident Object Fields

These are the fields available in the external incident object 'I' and for sorting incidents. For each field the equivalent Incident data access function is listed.

Incident Field Name	Function Call
getGroupingFlag	Incident::getGroupingFlag()
getMessages	Incident::getMessages()
getClosestMeshNode	Incident::getClosestMeshNode()
getHdl	Incident::getHdl()

Incident Field Name	Function Call
getSnd	Incident::getSnd()
getCondHdl	Incident::getCondHdl()
getPartition	Incident::getPartition()
getPhases	Incident::getPhases()
getCondStatus	Incident::getCondStatus()
getEvent	Incident::getEvent()
getPriority	Incident::getPriority()
getType	Incident::getType()
getClue	Incident::getClue()
getBitmask	Incident::getBitmask()
getTotalPriority	Incident::TotalPriority()
getLifeSupportCode	Incident::getLifeSupportCode()
getCriticalCustomer	Incident::getCriticalCustomer()
getCallCancelInd	Incident::getCallCancelInd()
getCallbackLateInd	Incident::getCallbackLateInd()
getTcode	Incident::getTcode()
getTroubleQueue	Incident::getTroubleQueue()
getShortDesc	Incident::getShortDesc()
getDrvInst	Incident::getDrvInst()
getCID	Incident::getCID()
getCustomerName	Incident::getCustomerName()
getCtype	Incident::getCtype()
getAddrBuilding	Incident::getAddrBuilding()
getAddrStreet	Incident::getAddrStreet()
getAddrCity	Incident::getAddrCity()
getOrderNumber	Incident::getOrderNumber()
getGenernalArea	Incident::getGeneralArea()
getMeterId	Incident::getMeterId()
getDeviceAlias	Incident::getDeviceAlias()
getNcg	Incident::getNcg()
getExternalId	Incident::getExternalId()
getInputTime	Incident::getInputTime()
getCallbackRequest	Incident::getCallbackRequest()

Incident Field Name	Function Call
getCallbackDataTime	Incident::getCallbackDataTime()
getOpComment	Incident::getOpComment()
getCustomerPhone	Incident::getCustomerPhone()
alternatePhone	Incident::alternatePhone()
userName	Incident::userName()
getXRef	Incident::getXRef()
getYRef	Incident::getYRef()
getExternallySent	Incident::getExternallySent()
getAONFlag	Incident::getAONFlag()
getROCFlag	Incident::getROCFlag()
active	Incident::active()

Permanent Order Object Fields

These are the fields that are always available in the external order object 'O'. These fields are read-only. The contents of each field are listed.

Incident Field Name	Contents
order	The order's Handle
event	The order's key event Handle
eventless	A boolean indicating that the order has lost all it's events. This is true for a period between the time the key event became non-existent and the time the order is completed, often as a result of an EventNonexistent trigger triggering an Output Document that completes the order.

Permanent Relationship Object Fields

These are the fields that are always available in the external relationship object 'O'. These fields are read-only. The contents of each field are listed.

Incident Field Name	Contents
relation	The relationship's Handle
event	The relationship's key (parent) event Handle
type	The relationship's type

Incident Field Name	Contents
Active	A boolean indicating that the relationship is active. This is false for a period between the time the relationship was deleted in Oracle Utilities Network Management System and the time the relation is deleted, often as a result of a RelationDeleted trigger triggering an Output Document that deletes the relation.

Chapter 8

ICCP Adapter

This chapter includes the following topics:

- **ICCP Adapter Overview**
- **ICCP Adapter Configuration**

ICCP Adapter Overview

The ICCP Adapter is an application that integrates the Oracle Utilities Network Management System with a remote SCADA system through the Inter-control Center Communications Protocol (ICCP). The ICCP Adapter employs the LiveData Server product provided by LiveData, Inc. LiveData Server is a mandatory component of the Oracle Utilities Network Management System ICCP Adapter and must be separately licensed from LiveData, Inc.

ICCP is a standard interface protocol that can be used with Oracle Utilities Network Management System to provide data exchange with remote and local SCADA systems. ICCP is also an international standard: International Electrotechnical Commission (IEC) Telecontrol Application Service Element 2 (TASE.2).

ICCP allows the exchange of real-time and historical power system monitoring and control data, including measured values, scheduling data, energy accounting data, and operator messages. Data exchange can occur between:

- Multiple control center Energy Management System (EMS) systems
- EMS and power plant DCS systems
- EMS and distribution SCADA systems
- EMS and other utility systems
- EMS/SCADA and substations

The ICCP standard consists of the following blocks:

Block	Description	Notes
Block 1	Basic Services	Available on Oracle Utilities Network Management System ICCP interface
Block 2	Extended Data Set Condition Monitoring	Available on Oracle Utilities Network Management System ICCP interface
Block 3	Blocked Transfers	
Block 4	Operator Stations	
Block 5	Device Control	Available on Oracle Utilities Network Management System ICCP interface
Block 6	Programs	
Block 7	Events	
Block 8	Accounts	
Block 9	Time Series	

ICCP Adapter Configuration

This section guides the user through configuration of the Oracle Utilities Network Management System ICCP Adapter. The following are assumed to be true before the adapter is installed:

- Oracle database access has been confirmed.
- Isis messaging bus has been installed and verified.
- Oracle Utilities Network Management System is installed and functional.
- LiveData Server is installed, functional, and licensed.

Configuring the ICCP Adapter requires:

- **Configuring the Adapter to Run as a System Service**
- **Populating the NMS Measurements Tables**

Configuring the Adapter to Run as a System Service

Configure the ICCP Adapter by updating the `$NMS_HOME/etc/system.dat` file to include the ICCP Adapter as a system service. There are three main sections where this service needs to be defined: the service, program and instance sections. See the `$CES_HOME/templates/system.dat.template` file for examples of how to configure the ICCP Adapter. Search for `IccpAdapter` and make sure those lines are uncommented. You must restart the system services in order for the ICCP Adapter to be properly monitored by SMService.

Below is an example of the program section in the system.dat file:

```
program  IccpAdapter   IccpAdapter  -prm_path /users/nms1/etc/
```

Note: It is assumed that the ICCP Adapter will reside on the same Unix or Linux server where the Oracle Utilities Network Management System services environment resides.

Command Line Options for ICCP Adapter

The command line for the ICCP Adapter provides the following options:

Command Line Option	What it does
-debug <level>	Sets the level of debug messages generated by the adapter. <level> is a positive number, or zero. The higher the number, the more information is displayed. If <level> is omitted, it defaults to a value of 0. Debug facilities can also be specified on the command line; for example: `-debug IA_RTP 3` could be used to specifiy level 3 debug for the IA_RTP debug facility.
-prm_path <IccpAdapter.prm path>	Sets the path of the IccpAdapter.prm parameter file location. This file is used to configure the operation of the ICCP adapter.
`-help`	Returns the available IccpAdapter startup parameters and definitions, then terminates.
-nodaemon	Runs in the foreground, used when running by hand.

IccpAdapter.prm

The IccpAdapter.prm file is used to configure the operation of the Oracle Utilities Network Management System ICCP Adapter. The default location for this file is the same as where the IccpAdapter binary is located (*i.e.*, `$CES_HOME/bin`) but it is generally configured to be in a different location by using the **`-prm_path <IccpAdapter.prm path>`** command line option. Lines in this file beginning with a ";" (semi-colon) are comments. Lines beginning with a "[" (left bracket) are block identifiers (markers). Fields marked as <Required> must be configured for proper operation and are generally site specific. See the IccpAdapter.prm.template file in the standard `$CES_HOME/templates` directory for an example IccpAdapter configuration file.

Fields in the IccpAdapter.prm File

Field name	Type	Default	Valid Values	Description
[IccpAdapter]	Marker			Used for generic configuration of program.
ServerHostname	IP address List – blank separated	<Required>	128.168.148.43 etc	The IP address(es) of the LiveData Server hostname(s) to connect to. It could be a blank separated list of IP address of several LiveData Servers. In case a failure of connection was detected by the ICCP Adapter with the current LiveData Server, it will traverse the ServerHostname list for the next LiveData Server to connect to.
Port	Integer	<Required>	[1..MAX_INT]	Blank separated list of TCP/IP port numbers that the ICCP Adapter will use for a connection attempt to a LiveData Server. Parallel to the ServerHostname, it could be a list of port numbers to use to connect to the corresponding LiveData server in ServerHostname. In case there was a failure of connection with the current LiveData Server, it would proceed to the next entry - in parallel with the next ServerHostname entry. 5002 is typical.
Period	Integer	10	[1..MAX_INT]	Time in seconds between periodic transfers of non-time critical data.
StatusUpdates	Integer	25	[1..MAX_INT]	The maximum number of status updates to be sent to DDService at one time.
ScadaId	Integer	1	[1..MAX_INT]	Identification number assigned to the SCADA in Oracle Utilities Network Management System with which the ICCP Adapter is communicating.
AnalogTolerance	Double	0.0F	[0.01..0.99]	Dead band for analog value updates. It is the required percent change from the last reported value to trigger an update.
Analogs	Boolean	F	[T, F]	Boolean value indicating use of the ANALOG_MEASUREMENTS table.
Digitals	Boolean	T	[T, F]	Boolean value indicating use of the DIGITAL_MEASUREMENTS table.
ReconnectPeriod	Integer	60	[0..MAX_INT]	Configurable duration of delay to wait after the LiveData Server instances failed in succession.
Controls	Boolean	F	[T, F]	Boolean value indicating use of the controls table for Block 5 functionality.
QualityCodeUseOn AssociationTimeOut	Integer	0	[0..MAX_INT]	Quality code that will be sent to DDService when the communication with LD server is lost. A valid QualityCode must be specified if this option is used.

Field name	Type	Default	Valid Values	Description
DisableStop	Boolean	F	[T, F]	Normally the adapter will accept and process a stop high level message. This option disables this feature. When this feature is enabled, the adapter will disregard a stop high level message.
DisableCOV	Boolean	F	[T, F]	Normally the adapter will process a COV update (one or more open and close sequences within a scan cycle – normally indicating one or more momentaries) and send it to DDService. This option disables this feature.
Vccs	Integer	<Required>	[1..MAX_INT]	The number of VCCs (Virtual Control Centers) that are configured in the LiveData Server.
IgnoreCritInterSysServFail	Boolean	F	[T, F]	Normally the adapter will stop if SMSserivce reports a critical service failure and not restart until services are recovered. This option disables this feature.
NoSwitchOpQualityMask	Integer	No Mask	[0..MAX_INT]	This parameter sets the quality codes that prevent switches from being operated. There is no effect on non-switch statuses.
PhaseEncodeSwitch	Boolean	F	[T, F]	If set to true, this will enable Iccp Adapter to interpret data discrete values as three-bit phase encoded statuses. [e.g., A = '001', B = '010', C = '100', etc.]
PseudoAlarms	Boolean	F	[T, F]	If set to 1, then this will set the pseudo flag for the switch entry to be sent to DDService. Generates pseudo (advisory) alarms for ICCP reported device ops rather than actually operating the switches in the Oracle Utilities Network Management System model.
SendTimeout	Integer	10	[0..MAX_INT]	Number of seconds to wait when attempting to connect ICCP Adapter to the LiveData server. If no connection is received, it will move to the next available LiveData server (if configured). Generally leave as the default.
DetachRead	Boolean	T	[T,F]	Detach the IccpAdapter internal thread that is reading the incoming RTP data stream from Isis. Generally leave as the default.

Field name	Type	Default	Valid Values	Description
DetachWrite	Boolean	F	[T,F]	Detach the IccpAdapter internal thread that is writing the outgoing RTP data stream from Isis. Generally leave as the default
DetachHeartbeat	Boolean	F	[T,F]	Detach the IccpAdapter internal thread that is sending outgoing RTP data stream heartbeat requests from Isis. Generally leave as the default
[VCC#]	Marker			E.g., [VCC1]. Provides additional information for each VCC (Virtual Control Center).
AssociationAddress	Integer	<Required>	[1..MAX_INT]	RTP address in LiveData Server for watching and controlling this VCCs association status
TransferSetAddress	Integer	<Required>	[1..MAX_INT]	RTP address in LiveData Server for controlling the use of configured ICCP transfer sets
NumTransferSets	Integer	<Required>	[1..MAX_INT]	The total number of transfer sets that are available for use in the VCC. Number must be a multiple of 16.
AssociationName	String	Vcc Label	[a..z, A..Z, 0..9]	The name of the ICCP Association.
AssociationRestartTime	Integer	30	[1..MAX_INT]	Seconds allowed for restart before the association is considered failed.
TransferSetRestartPeriod	Integer	30	[1..MAX_INT]	Seconds allowed to restart transferset before the restart is considered failed and no additional restart attempts will be made.
TransferSetFailCountReset	Integer	60	[1..MAX_INT]	The number of fail count to be exhausted before marking the transfer set as not alive.
MaxTransferSetRestarts	Integer	10	[1..MAX_INT]	Maximum number of restart for transfer set.
TransferSetControlMask	String	<Required>	[T, F]	Transfer set control mask for the transfer set to be sent to LiveData Server. One T/F flag for each TransferSet. String length must be a multiple of 16. Example with one TransferSet enabled: "FTTTTTTTTTTTTTTT"
[ValidityQuality]	Marker			Assign an Oracle Utilities Network Management System quality to ICCP Validity Quality values
Valid	Integer	0	2**n (n=11->31)	The value is valid. This is the default (normal) value should virtually always be 0.
Held	Integer	0	2**n (n=11->31)	Previous data value has been held over. Interpretation is local.

Field name	Type	Default	Valid Values	Description
Suspect	Integer	0	2**n (n=11->31)	Data value is questionable. Interpretation is local.
Notvalid	Integer	0	2**n (n=11->31)	The value is not valid.
[CurrentSourceQuality]	Marker			Assign an Oracle Utilities Network Management System quality to ICCP Current Source Quality values.
Telemetered	Integer	0	2**n (n=11->31)	Value was received from a telemetered site. This is the default (normal) value should virtually always be 0.
Calculated	Integer	0	2**n (n=11->31)	Value was calculated based on other data.
Entered	Integer	0	2**n (n=11->31)	Value was entered manually.
Estimated	Integer	0	2**n (n=11->31)	Value was estimated (State Estimator, etc.).
[NormalValueQuality]	Marker			Assign an Oracle Utilities Network Management System quality to ICCP Normal Value Quality values.
Normal	Integer	0	2**n (n=11->31)	The point value is that which has been configured as normal for the point. This is the default (normal) value should virtually always be 0.
Abnormal	Integer	0	2**n (n=11->31)	The point value is *not* that which has been configured as normal for the point.
[TimeStampQuality]	Marker			Assign an Oracle Utilities Network Management System quality to ICCP Timestamp Quality values
Valid	Integer	0	2**n (n=11->31)	Current value of the TimeStamp attribute contains the time stamp of when the value was last changed. This is the default (normal) value should virtually always be 0.
Invalid	Integer	0	2**n (n=1->31)	Current value of the TimeStamp attribute contains the time stamp other than when the value was last changed.
[SwitchStatusQuality]	Marker			Assign an Oracle Utilties Network Management System quality to the non-open/close statuses that can be returned in the two-bit ICCP status field. ICCP “open” is generally (1) and “closed” is (2).
Between	Integer	262144	2**n (n=11->31)	Qualithy code to set if the two bit ICCP switch status is reported as “between” (0).
Invalid	Integer	524288	2**n (n=11->31)	Qualithy code to set if the two bit ICCP switch status is reported as “invalid” (3).

Sample IccpAdapter.prm Configuration File

```
[IccpAdapter]
Period=5
ScadaId=1
Analogs=0
AnalogTolerance=.0001
Digitals=1
Controls=0
Port=5002
QualityCodeUseOnAssociationTimeOut=16384
Vccs=1
DisableCOV=0
[VCC1]
AssociationAddress=10
TransferSetAddress=20
NumTransferSets=16
[ValidityQuality]
Valid=
Held=
Suspect=
Notvalid=1048576
[CurrentSourceQuality]
Telemetered=
Calculated=
Entered=
Estimated=2097152
[NormalValueQuality]
Normal=
Abnormal=
[TimeStampQuality]
Valid=
Invalid=
```

Quality Codes

The IccpAdapter.prm file enables ICCP quality codes to be translated into Oracle Utilities Network Management System quality codes. In the simplest (and default) configuration, all of the ICCP quality codes (except the `Between` and `Invalid SwitchStatusQuality` codes, which need to be defined to ensure proper operation) are assigned to the 'normal' Oracle Utilities Network Management System quality code (0).

> **Note**: Oracle Utilities Network Management System quality codes are always single bit values. Therefore, the only valid value for configuration is 0 or a proper value of 2^nth power where n=0->31. The **Quality Rules Table** on page 8-11 table lists all the valid user-defined quality codes in Oracle Utilities Network Management System.

If none of the predefined quality codes are applicable, then a new code must be created. The following steps accomplish this:

- Choose an ICCP quality listed in the IccpAdapter.prm.
- Check the **Quality Rules Table** to see which values have already been assigned to qualities.
- Assign one of the values listed below to the ICCP quality and enter it in the **Quality Rules Table**.
- Locate the quality in the IccpAdapter.prm file and enter the assigned value for it.

The assigned value must be the decimal representation of 32 bits, where no more than one bit has a value of 1. For example, if the bit position is 11, use the number 2048. The following list contains the decimal values that may be assigned to new qualities: 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, 2147483648.

Values of 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 may not be assigned as codes for new qualities because they are already defined and used within Oracle Utilities Network Management System. The 'normal' Oracle Utilities Network Management System quality code is 0.

The adapter reads the IccpAdapter.prm file only during startup. If the quality code is added when the adapter is running, you must restart the adapter in order for it to recognize the new quality code.

High Level Messages

The ICCP Adapter can be dynamically controlled from Oracle Utilities Network Management System by using high-level messages. They can be used any time while running the Oracle Utilities Network Management System ICCP Adapter. The following high-level messages can be used:

- `stop`

  Disconnect from the LiveData Server and stop the Oracle Utilities Network Management System ICCP Adapter.

- `report`

  Empty message to determine how many Oracle Utilities Network Management System ICCP Adapters are running.

- `debug [on | off | #]`

  Turn on/off debug, or set it to a specific level. On is equivalent to 1, off is 0. Level can be any integer value no less than 0.

- `debug <facility> #`

  Turn facility specific debug on/off. For example, to turn IA_RTP debug on to level 3:

  ```
  Action any.IccpAdapter debug IA_RTP 3
  ```

  To turn off:

  ```
  Action any.IccpAdapter debug IA_RTP 0
  ```

  Check ICCP Adapter specific log file for other facilities specific to this adapter process.

- `demote`

  Causes the Oracle Utilities Network Management System ICCP Adapter currently in control to relinquish control.

Use IccpAdapterService with high-level messages for the Oracle Utilities Network Management System ICCP Adapter. For example:

```
Action any.IccpAdapter report
```

Populating the NMS Measurements Tables

ICCP points must first be mapped to devices in the Oracle Utilities Network Management System model before sending SCADA updates to Oracle Utilities Network Management System. These ICCP points are placed in DIGITAL_MEASUREMENTS and ANALOG_MEASUREMENTS tables of the Oracle Utilities Network Management System database. A process needs to be formalized to create and maintain this data. This process often depends on customer specific mechanisms used to maintain the SCADA side of the ICCP interface. As a result this process generally needs to be formalized by LiveData and the customer - potentially with help from NMS consulting.

Required NMS Data from LiveData

The following data items are required to be populated in the NMS measurements tables:

- **ICCP Name**
- **ICCP Type**
- **Attribute**
- **NMSDeviceID**

ICCP Name

- ICCP Name has to be unique.
- It is recommended for the name to be composed of alpha-numeric characters and underscore
- It is recommended for the first character of the ICCP Name to be a letter
- There should be no space, no periods and no dashes in the ICCP name.

ICCP Type

Below is a list of supported ICCP Types. Please take note of the underscore.

- Data_State
- Data_StateQ
- Data_StateQTimeTag
- Data_StateExtended
- Data_Real
- Data_RealQ
- Data_RealQTimeTag
- Data_RealExtended
- Data_Discrete
- Data_DiscreteQTimeTag
- Data_DiscreteExtended

Attribute

- The attribute should have a corresponding entry in the ATTRIBUTES table, specifically, in the NAME field in the Oracle Utilities Network Management System database. Take note that entries under the attributes column of the flat file needs to exactly match the entries in the NAME field of the ATTRIBUTES table, taking into consideration case sensitivity, underscores, etc.

NMSDeviceID

- This is the ID of the SCADA device. This ID should match a unique attribute or device name in NMS that will allow the measurement table population process to grab the appropriate NMS device handle (h_cls and h_idx) of the SCADA device. Tables that could be used in NMS to reference SCADA devices handles could be SCADAMAP, ALIAS_MAPPING, or a model managed device attribute table such as ATT_SWITCH. Take note that matching should put into consideration several aspects like case sensitivity, special characters such as hyphens, underscores, etc.

Information Model - Database Schema

Quality Rules Table

This database table will define the quality codes that may be used for analog and digital values. This table defines the meaning of each bit in the quality codes for SCADA measurements.

Column Name	Data Type	Size	Description	Values
PRIORITY	NUMBER		Ranking priority of the quality code	Priority code, specifies relative importance of this quality bit over other quality bits
VALUE	NUMBER		Bit value used for the quality code change.	For Phase 1: 2048=No Data 4096=Old Data
STRING	VARCHAR2	3	Description of the quality code, which is displayed next to the value of the measurement when a quality exists for a measurement change.	The actual character string displayed next to the device when viewed via the Viewer
DESCRIPTION	VARCHAR2	128	Descriptive string	Any text string-usually the action taken from the SCADA Summary
COLOR	NUMBER		Designates which color is used in the Viewer to display the measurement when a particular quality bit is set. Integer value for the color associated to the quality code change to be displayed	The integers are mapped to the pre-allocated colors documented in separate application file.
LOCATION	NUMBER		Location of symbol in relation to the device associated with the value. only used if a symbol is defined for the quality code as apposed to just a color for a quality change	1-9; 5 overrides the device symbol
SYMBOL	NUMBER		The symbol used to display the value. only used if a symbol is defined for the quality code as apposed to just a color for a quality change	Valid Symbol Identification Number defined in <project>_SYMBOLS.sym. 0 if defining a text symbol.
OFF_NOMINAL	VARCHAR2	1	Whether or not the value is off-nominal	Y or N

Note: If multiple bits in the quality code are set, then the color of the measurement text in the Viewer is determined by the color of the lowest order bit that is set in the quality code.

Digital and Analog Measurements Tables

These tables define digital and analog measurements as used by Oracle Utilities Network Management System. They also manage the existence of the measurement and linkage information. Manually overridden values may also be stored here.

The Oracle Utilities Network Management System ICCP Adapter communicates dynamic information to the Oracle Utilities Network Management System services. The services will cache measurements defined by this table. Population is dependent upon customer-supplied information.

Digital_Measurements Table

Column Name	Data Type	Size	Description	Values
H_CLS	NUMBER		Class component of handle	Valid object class
H_IDX	NUMBER		Index component of handle	>0
PARTITION	NUMBER		Partition number, index component of partition handle	Valid partition or 0 for multi-partition objects
ATTRIBUTE	NUMBER		Attribute number which identifies measurement type	Valid attribute number
TTL	NUMBER		Setting for displaying measurement value in the Viewer or not.	1 or 0. 1=yes
LIMIT_GROUP_ID	NUMBER		Limit group ID	Customer defined
RTI_ALIAS	VARCHAR2	128	Alias to be used in communications between the Oracle Utilities Network Management System ICCP Adapter and LiveData Server	Alphanumeric
SCADA_ID	NUMBER		SCADA host ID	0 (not a SCADA device), 1 (SCADA 1), 2 (SCADA 2)...
RTU_ID	VARCHAR2	32	SCADA RTU ID	String (optional)
QUALITY	NUMBER		Measurement quality code	Bit mask of quality codes
VALUE	NUMBER		Measurement/entered value	Entered value
UPDATE_FLAG	NUMBER		Manual replace flag	1=true, 0=false
ICCP_OBJECT	VARCHAR2	32	ICCP Object type of the telemetered value	Alphanumeric
DISPLAY_ID	VARCHAR2	64		
NORMAL_STATE	NUMBER			
CONTROLLABLE	VARCHAR2	1		
ACTIVE	VARCHAR2	1	Active flag for patch management, indicates whether the row is active within the model	Y(yes=active), N(no=inactive), A(local add=active), D(localdelete=inactive), R(locally removed, dependent=inactive)
SOURCE	VARCHAR2	33	Source of the measurement.	any character string.

Column Name	Data Type	Size	Description	Values
COMMENTS	VARCHAR2	512	Operator-entered comments	any character string.
OFF_NOMINAL_TIME	DATE			

Analog_Measurements Table

Column Name	Data Type	Size	Description	Values
H_CLS	NUMBER		Class component of handle	Valid object class
H_IDX	NUMBER		Index component of handle	>0
PARTITION	NUMBER		Partition number, index component of partition handle	Valid partition or 0 for multi-partition objects
ATTRIBUTE	NUMBER		Attribute number which identifies measurement type	Valid attribute number
TTL	NUMBER		Setting for displaying measurement value in the Viewer or not.	1 or 0. 1=yes
LIMIT_GROUP_ID	NUMBER		Limit group ID	Customer defined
RTI_ALIAS	VARCHAR2	128	Alias to be used in communications between the Oracle Utilities Network Management System ICCP Adapter and LiveData Server	Alphanumeric
SCADA_ID	NUMBER		SCADA host ID	0 (not a SCADA device), 1 (SCADA 1), 2 (SCADA 2)...
RTU_ID	VARCHAR2	32	SCADA RTU ID	String (optional)
QUALITY	NUMBER		Measurement quality code	Bit mask of quality codes
VALUE	FLOAT	126	Measurement/entered value	Entered value
UPDATE_FLAG	NUMBER		Manual replace flag	1=true, 0=false
ICCP_OBJECT	VARCHAR2	32	ICCP Object type of the telemetered value	Alphanumeric
DISPLAY_ID	VARCHAR2	64		
CONTROLLABLE	VARCHAR2	1		
ACTIVE	VARCHAR2	1	Active flag for patch management, indicates whether the row is active within the model	Y(yes=active), N(no=inactive),A(local add=active), D(local delete=inactive), R(locally removed, dependent=inactive)
SOURCE	VARCHAR2	33	Source of the measurement.	any character string.
COMMENTS	VARCHAR2	512	Operator-entered comments	any character string.

Column Name	Data Type	Size	Description	Values
OFF_NOMINAL_TIME	DATE			

Controls Table

This table defines control actions as used by Oracle Utilities Network Management System. Population is dependent upon customer-supplied information. The information to be contained in this table is generated by the Auto Configuration Program.

Column Name	Data Type	Size	Description	Values
DEVICE_CLASS	NUMBER		Class component of device handle.	Valid object class
DEVICE_ID	NUMBER		Index component of device handle.	>0
CTLTYPE_ID	NUMBER		The Control Action ID number associated to the action.	Valid control action ID. 1 (OPEN), 2 (CLOSE)…
CONTROL_ID	NUMBER		Part of unique key to identify each control action for a single device.	0..N, based on the number of control actions defined for the device.
EXP_STATE	NUMBER		Expected return value.	Numeric
ATTRIBUTE	NUMBER		If non-zero, attribute number which identifies measurement type.	0 (Ignore) or Valid attribute number
RTI_ALIAS	VARCHAR2	128	Alias to be used in communications between the Oracle Utilities Network Management System ICCP Adapter and LiveData Server.	Alphanumeric
TIMEOUT	NUMBER		SCADA timeout for this device.	0 = No Timeout, >0 = timeout is seconds.
NOT_IN_SERVICE	VARCHAR2	1	Not in service flag.	N (In Service), Y (Not in Service)
SCADA_ID	NUMBER		SCADA server ID.	0 (not a SCADA device), 1 (SCADA 1), 2 (SCADA 2)...
RTU_ID	VARCHAR2	32	SCADA RTU ID.	String (optional)
ACTIVE	VARCHAR2	1	Active flag for patch management; indicates whether the row is active within the model.	Y (yes=active), N (no=inactive), A (local add=active), D (local delete=inactive), R (locally removed, dependent=inactive)

Chapter 9

MultiSpeak Adapter

The Oracle Utilities Network Management System MultiSpeak Adapter provides the ability to request and receive meter status information from an Automated Meter Reading (AMR) system and provides the ability to receive crew location information from an Automated Vehicle Location (AVL) system. The interface uses communication protocols as defined in MultiSpeak Version 4.0 Web Services specification. (SOAP protocol version 1.1 is used unless otherwise noted.) HTTPS protocol is used as transport mechanism (plain HTTP is not supported). It allows Oracle Utilities Network Management System to communicate securely with any MultiSpeak-compliant AMR or AVL system.

The Oracle Utilities Network Management System MultiSpeak Adapter is implemented as a Java application, running on the Oracle WebLogic Server or the JBoss Application Server platform.

Please read through this chapter thoroughly before beginning your product installation.

Installation

This chapter describes how you install the Oracle Utilities Network Management System MultiSpeak Adapter, including:

- Manual Installation Instructions for Oracle WebLogic Server
- Manual Installation Instructions for JBoss Application Server

> **Note**: It is assumed that the Oracle Utilities Network Management System Web Gateway component has already been installed. Refer to Oracle Utilities Network Management System Installation Guide for complete instructions on how to install and configure the Oracle Utilities Network Management System Web Gateway.

Installation Overview

The Oracle Utilities Network Management System MultiSpeak Adapter is delivered as a single file: nms-amr.ear.base.

This file is installed into the $CES_HOME/dist/install/deploy directory during the Oracle Utilities Network Management System installation process. The *nms-install-config* script has to be used in order to apply adapter configuration changes and create the *nms-amr.ear* file, which can be deployed to the Java Application server (see **Software Configuration** on page 9-3 for configuration instructions).

Adapter Installation Instructions for Oracle WebLogic Server

The Oracle WebLogic Server Administration Console enables you to deploy the Utilities Network Management System MultiSpeak Adapter as follows:

1. Log into the WebLogic Server Administration Console:

   ```
   http://<weblogic machine name>:7001/console/
   ```

   where <weblogic machine name> is the machine name where WebLogic Server is installed.
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. In the right pane, click **Install**.
5. In the Install Application Assistant, locate the nms-amr.ear file.
6. Click **Next**.
7. Specify that you want to target the installation as an application.
8. Click **Next**.
9. Select the servers and/or clusters 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.
10. Click **Next**.
11. Set the deployed name of the application to be *nms-amr*.
12. Click **Next**.
13. Review the configuration settings you have specified, and click **Finish** to complete the installation.

Adapter Installation Instructions for JBoss Application Server

To install the Utilities Network Management System MultiSpeak Adapter in the JBoss Application Server, do the following:

- Copy the nms-amr.ear file to the following directory:

  ```
  <JBoss installation directory>/server/default/deploy
  ```

Software Configuration

Configuration for the Oracle Utilities Network Management System MultiSpeak Adapter comes from the following sources:

- AMRInterface.properties file
- AVLInterface.properties file
- Oracle Utilities Network Management System configuration rules

Changing Configuration Files

The AMRInterface.properties and AVLInterface.properties files are deployed into the application server as part of the nms_amr.ear file. Follow the instructions from the Java Application Configuration chapter of the Oracle Utilities Network Management System Configuration Guide to modify these configuration files for customer-specific changes and redeployment into the application server. Definitions of the contents of the property files are provided in the sections that follow.

> **Note**: In order for changes to the property files to take effect, the interface has to be redeployed or the application server has to be restarted (see **Installation Overview** on page 9-1 for deployment instructions).

AMRInterface.properties file

The AMRInterface.properties file contains configuration for the AMR component of the Oracle Utilities Network Management System MultiSpeak Adapter.

The following table describes the general configuration properties.

Property	Description
config.credentials	Absolute path to the file containing user credentials the adapter will use to communicate with Oracle Utilities Network Management System. This property is required.
config.amr_vendor	AMR vendor. Supported AMR vendors: This property is required. Default = multispeak (any Multispeak-compliant AMR System)
config.ping_request_interval	Time interval (in minutes) between subsequent meter ping requests to the AMR system. Default = 3
config.enabled	Enables AMR processing. Default = true

The following table describes configuration properties specific to a particular AMR vendor. This could be any MultiSpeak-compliant AMR system.

Property	Description
multispeak.meter_status.<*external status*>	This property configures mapping between external (MultiSpeak) and internal meter status values. Valid values are: ON - meter is energized OFF - meter is deenergized UNKNOWN - external meter status has no configured mapping, Examples: multispeak.meter_status.Outage=OFF multispeak.meter_status.PowerOff=OFF multispeak.meter_status.PowerOn=ON multispeak.meter_status.Restoration=ON multispeak.meter_status.Instantaneous=UNKNOWN multispeak.meter_status.NoResponse=UNKNOWN multispeak.meter_status.Inferred=UNKNOWN
multispeak.od_oa.url	This property configures the URL of the AMR system web service. Default: https://localhost/multispeak
multispeak.od_oa.username	Username to use when connecting to the AMR system web service. Default: empty string
multispeak.od_oa.password	Password to use when connecting to the AMR system web service. Default: empty string
multispeak.od_oa.header.<attribute>	Used to set the values for MultiSpeak header attributes. For example, the following would set the MultiSpeak header attribute “Company” to the value “Oracle”: multispeak.od_oa.header.Company=Oracle
multispeak.od_oa.soap12	Indicates the SOAP protocol version to use for communicating with the AMR/AMI system. If true, version 1.2 will be used. Otherwise, version 1.1 will be used. Default: false (SOAP version 1.1 is used)
multispeak.max_ping_attempts	Maximum number of attempts to ping a meter. Default: 3
multispeak.ping_attempt_interval	Amount of time (in seconds) to wait for reply from the AMR system before resending meter ping request. Default: 60

AVLInterface.properties file

The AVLInterface.properties file contains configuration for the AVL component of the Oracle Utilities Network Management System MultiSpeak Adapter. It includes configuration for converting crew location information received from the AVL system into the Oracle Utilities Network Management System coordinate system. At least two reference points are required for coordinate conversion to work.

The AVL properties are described in the following table:

Property	Description
config.credentials	Absolute path to the file containing user credentials the adapter will use to communicate with Oracle Utilities Network Management System during initialization process. This property is required.
avl.num_reference_points	Number of configured reference points
avl.reference_point<N>.x	X coordinate of the reference point N in the Oracle Utilities Network Management System coordinate system
avl.reference_point<N>.y	Y coordinate of the reference point N in the Oracle Utilities Network Management System coordinate system
avl.reference_point<N>.longitude	Geographic longitude of the reference point N
avl.reference_point<N>.latitude	Geographic latitude of the reference point N
avl.xy_scale	Number of decimal points to use when rounding X/Y coordinates
avl.lat_long_scale	Number of decimal points for rounding longitude/ latitude coordinates
config.enabled	Enables AVL processing Default: true

.The following illustration shows a sample AVL configuration.

```
#
# Enable AVL processing
#
config.enabled=true

# Coordinate system mapping for AVL
#
avl.num_reference_points=2
avl.reference_point1.x=0.0
avl.reference_point1.y=0.0
avl.reference_point1.longitude=-93.508
avl.reference_point1.latitude=44.996
avl.reference_point2.x=100.0
avl.reference_point2.y=100.0
avl.reference_pount2.longitude=-93.464
avl.reference_point2.latitude=45.015
avl.xy_scale=2
avl.lat_long_scale=4
```

Credentials Files

Credentials files are used to configure usernames and passwords to be used by the parts of the adapter that communicate with the Oracle Utilities Network Management System.

Credentials files should only be readably by the operating system account under which application server is running.

The format of a credentials file is described in the following table:

Property	Description
nms.username	Valid NMS username.
nms.password	NMS user password

The following illustration shows a sample credentials file.

```
#
nms.username=amr
nms.password=amr-user-password
```

Oracle Utilities Network Management System AMR Configuration Rules

Below is the list of configuration rules in the Oracle Utilities Network Management System, which control AMR-related functionality. These rules are not directly used by the Oracle Utilities Network Management System MultiSpeak Adapter.

amrDeadband

Specifies the deadband for incoming AMR-reported incidents. If an incident is within the deadband from the outage restoration time, it will be discarded. This value must not be more than 86400 seconds (1 day).

amrInterfacesEnabled

This rule enables AMR processing in JMService. Its value indicates the AMR processing types that are available. AMR processing is disabled if this rule is set to 0 (default value).

Available types of AMR processing:

- 1 - Outage detection
- 2 - PSO verification
- 4 - PDO verification
- 8 - Restoration verification
- 16 - Manual AMR processing

The rule value is a bitmask, which allows any combination of AMR processing types to be enabled. For example, if the rule is set to 9 then outage detection and restoration verification will be enabled.

meterOffThreshold

Maximum probability of meter having power when meter is still assumed to be "off". Default value is 0.

meterOffTroubleCode

Trouble code to be used when a call should be created because of information received from AMR system.

meterOnThreshold

Minimum probability of meter having power when meter is still assumed to be "on". Default value is 100.

meterQueryThreshold

This parameter is used to determine if a meter can be queried when an active request exists. When a new request is made, existing requests will be evaluated to see if any contain meter(s) from the new request. If a match is found, and the difference between the time the request was received and current time is less than value of this rule in seconds, that meter will be rejected from the new request. If set to -1 (default value), this rule will not be enforced.

meterPingPercentage

This parameter governs the percentage of meters to ping for an AMR action. When set to 100, it will ping all AMR meters downstream from the outage device. When set to -1, it will ping one AMR meter on each SND. When set to any other number between 1 and 99, JMService will attempt to ping the specified percentage of downstream meters. Default value of this rule is 100.

momentaryAmrDeadband

Specifies the momentary deadband for incoming AMR-reported incidents. If an incident is within the deadband from a begin time of a momentary outage, it will be discarded. This value must not be more than 86400 seconds (1 day).

powerOutWaitPeriod

Wait period for AMR-reported power outs. System will hold onto AMR-reported power out messages for the specified period of time before making determination if the power out should be discarded, grouped into an existing outage, or if it should create a new outage.

useMeterTimeForDetection

This configuration rule determines if meter read time reported by AMR system should be used as call time for incidents created by outage detection functionality.

Valid values:

- 0 - Meter read time will not be used, instead current system time will be used (this is the default behavior)
- 1 - Meter read time will be used

suppressAmrIncidents

This rule controls how AMR-reported power outs, which occur downstream of a non-momentary device outage, are processed. If this rule is set to "yes" then such power outs are discarded; otherwise, they are grouped into the device outage.

useMeterTimeForRestoration

This configuration rule determines if meter read time reported by AMR system should be used to adjust outage restoration time as part of outage restoration verification functionality.

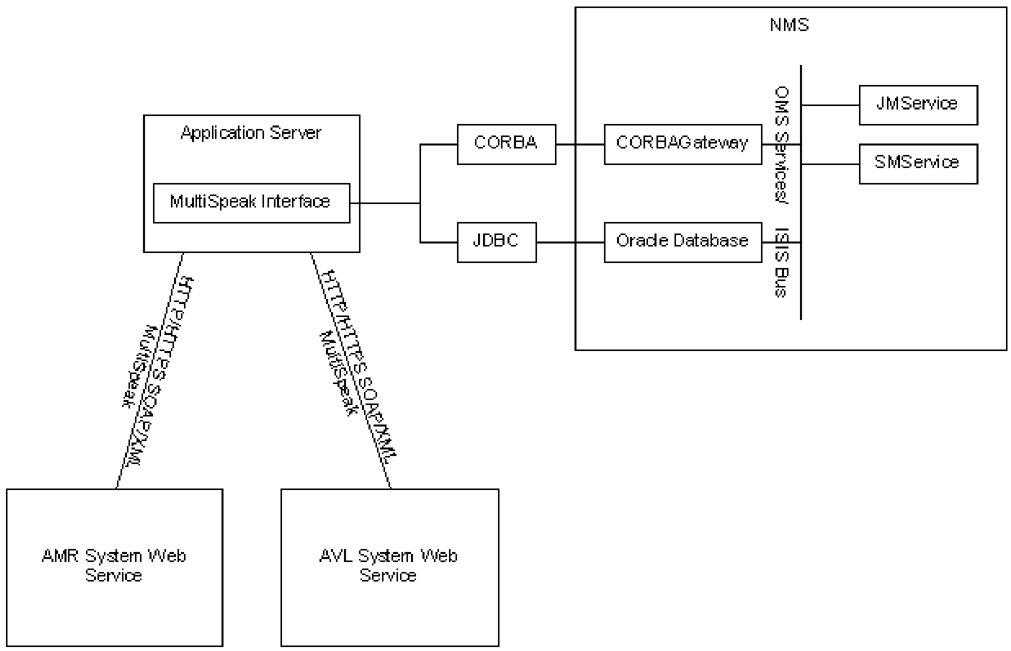
Valid values:

- 0 - outage restoration time will not be modified
- 1 - outage restoration time will be updated with the latest meter read time amongst the meters which reported power on for the restored outage (this is the default behavior)

Adapter Interface Communication Overview

The Oracle Utilities Network Management System MultiSpeak Adapter provides a reliable and configurable way of connecting MultiSpeak-compliant AMR and AVL systems to the Oracle Utilities Network Management System. The interface connects to the AMR systems by use of MultiSpeak compliant SOAP/XML calls over the HTTPS protocol. The interface connects to the Oracle Utilities Network Management System through direct database access using JDBC and by use of a CORBA connection through the Oracle Utilities Network Management System Web Gateway.

Interface Communication Overview



Adapter Design

Supported Data Flows

Oracle Utilities Network Management System MultiSpeak Adapter supports following data flows described in the MultiSpeak Web Services Version 4.0 specification.

Oracle Utilities Network Management System to an AMR system:

- **InitiateOutageDetectionEventRequest**

  Oracle Utilities Network Management System requests meter status information from the AMR system.

An AMR system to Oracle Utilities Network Management System:

- **ODEventNotification**

  AMR system reports meter status information to the Oracle Utilities Network Management System.

An AVL system to Oracle Utilities Network Management System:

- **AVLChangedNotification**

  AVL system reports crew location information to the Oracle Utilities Network Management System.

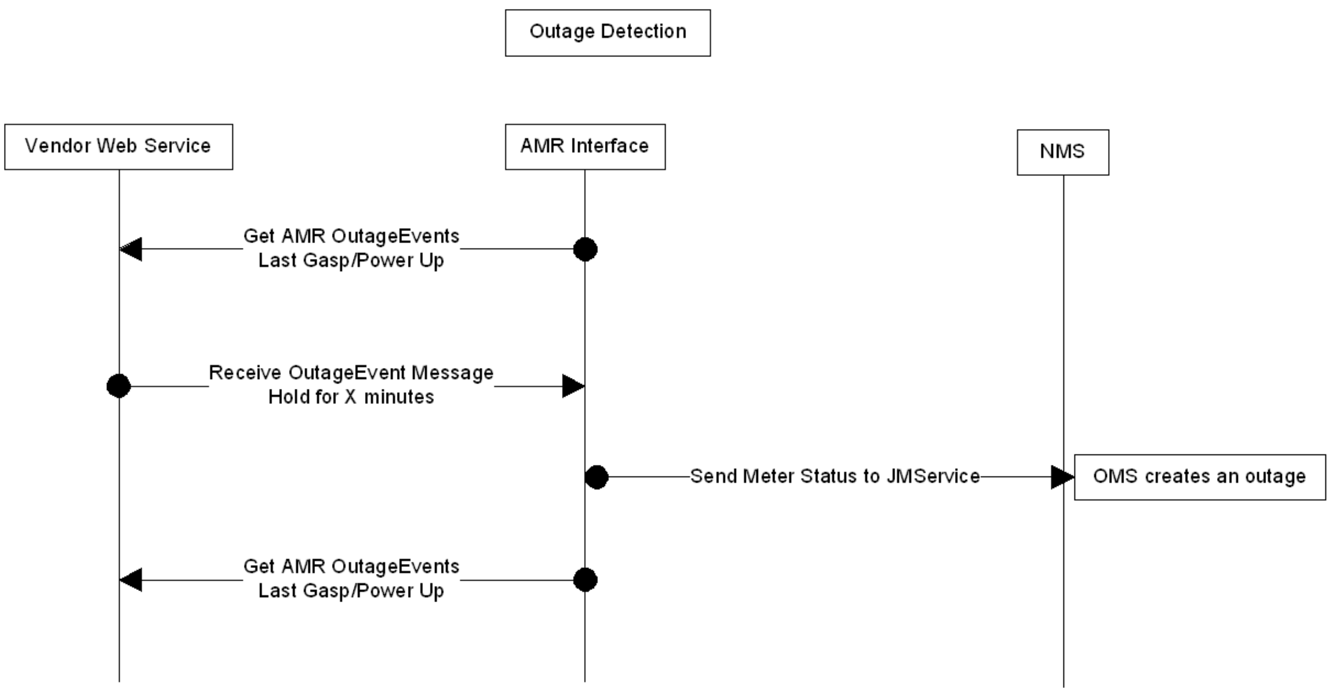
Incoming requests (ODEventNotification and AVLChangedNotification) are authenticated against list of valid Oracle Utilities Network Management System users. Username and password has to be provided in the header of each incoming MultiSpeak message.

AMR Business Processes

This section describes the utility business processes related to AMR that can be supported through the Oracle Utilities Network Management System MultiSpeak Adapter.

Outage Detection

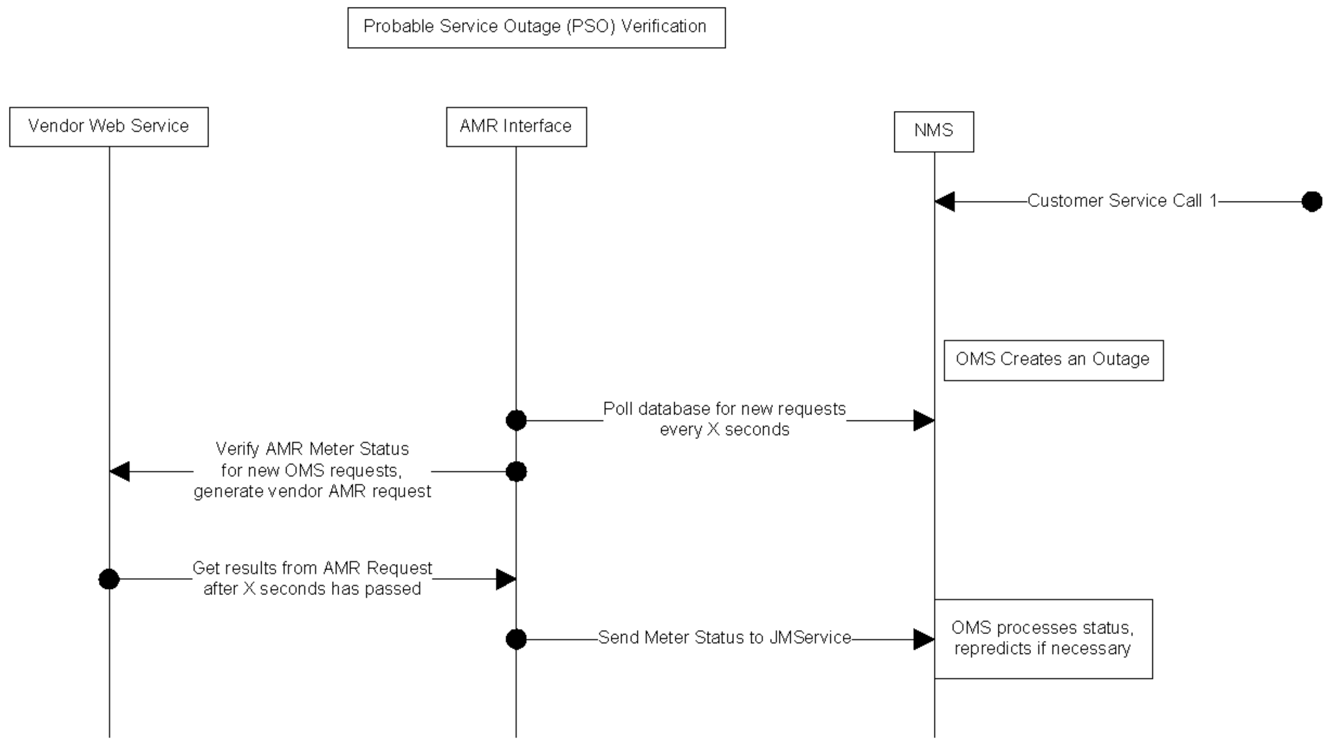
The vendor AMR system detects no power for a meter, either because of a "last gasp" meter message or from scheduled meter polling. A "power out" call is submitted to Oracle Utilities Network Management System, which generates a probable outage event.



PSO Verification

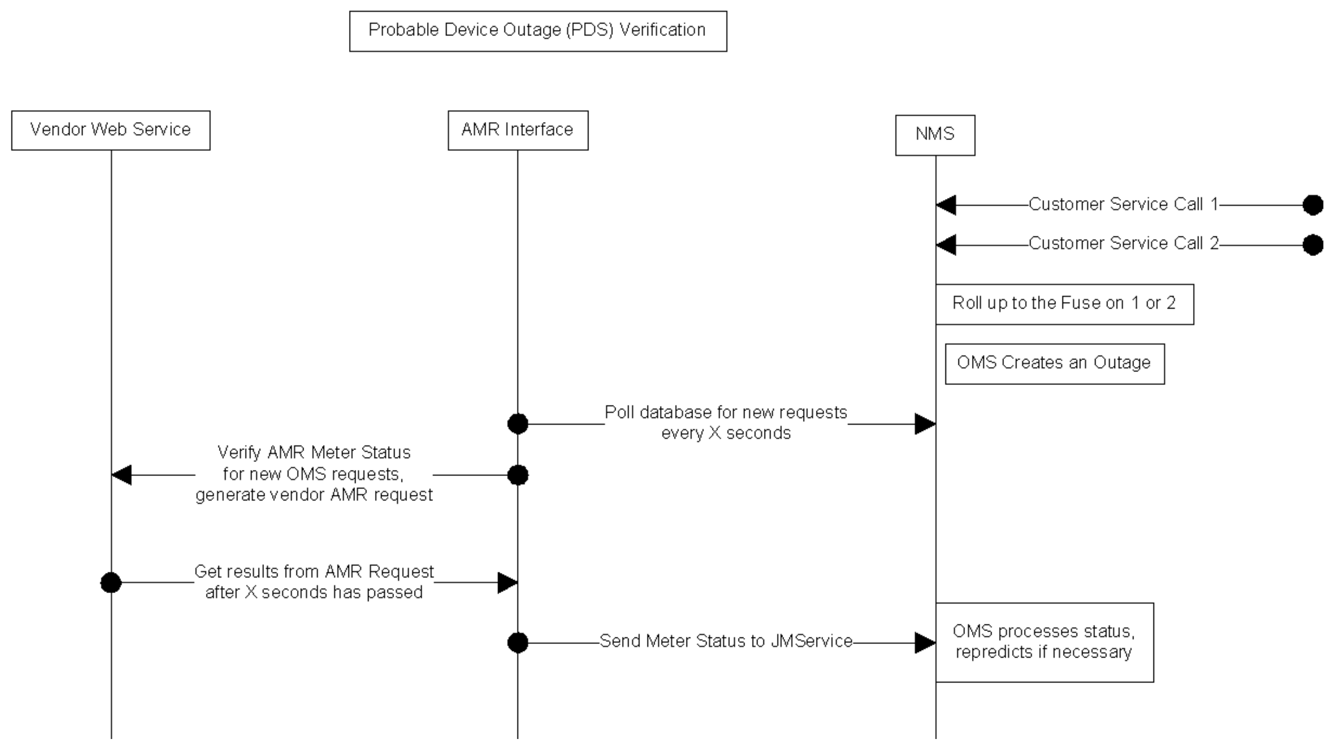
One customer call is received, generating a probable service outage in Oracle Utilities Network Management System. The Oracle Utilities Network Management System MultiSpeak Adapter is notified of the new probable outage, and the customer meter is pinged to verify power status.

If the meter reports that the power is still on, then we have conflicting information from the customer and the meter, so the outage predication engine will set the status of this event to Verify. At this point, we believe that there is no outage, but that the customer has a problem, such as a blown fuse, within his home. This event must be resolved by a customer service representative contacting the caller to explain the situation to them.



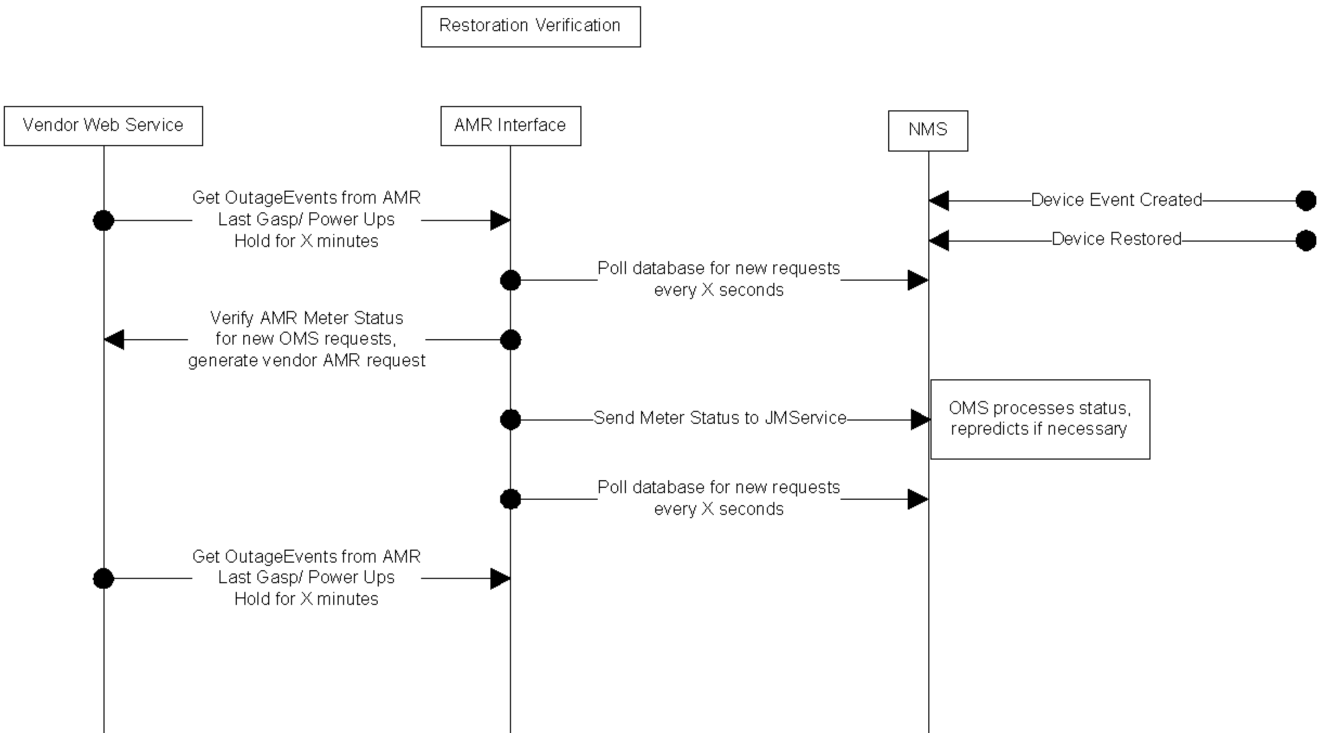
PDO Verification

Several customer calls are received, which are submitted into the Oracle Utilities Network Management System. The resulting probably outage rolls up to a device. The list of affected AMR customers is provided to the Oracle Utilities Network Management System MultiSpeak Adapter by the Oracle Utilities Network Management System outage prediction engine. The interface submits meter status requests to the AMR for any of the affected meters from which it has not already received a last gasp message. The received meter statuses are sent back to the prediction engine and the predicted outage device may change by moving downstream.



Restoration Verification

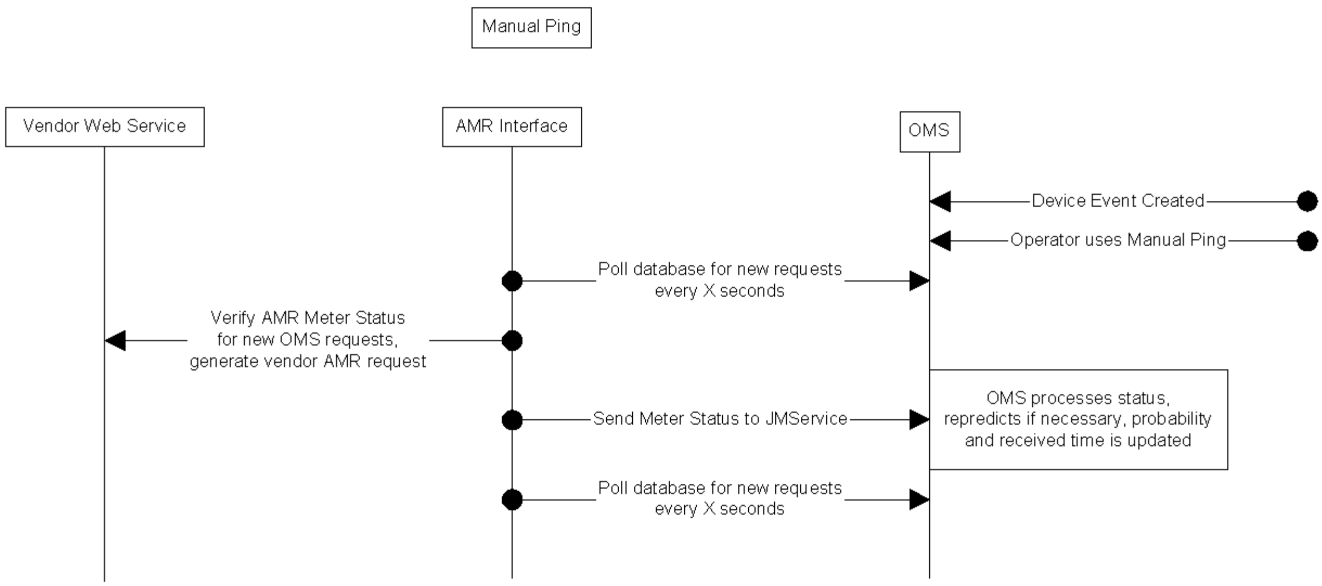
An outage event is restored in Oracle Utilities Network Management System, and a list of affected meters is provided by the outage prediction engine to the Oracle Utilities Network Management System MultiSpeak Adapter. The interface submits meter status requests to the AMR for any of the affected meters from which it has not received a “power up” message. The results are passed back to the Oracle Utilities Network Management System and the periodic cycle for getting outage events continues. The received meter statuses are sent back to the prediction engine. A power out status will result in another outage call and a nested outage that still needs restoration.



Manual Ping

In the diagram below, please note that the number indicates the sequence of actions:

1. The operator or system has chosen a device to “Ping”.
2. Information about the new ping request is stored in the database.
3. The AMR application notes the new ping request and verifies the device.
4. A response is received from the meter.
5. The database is updated with some information about the request response. Oracle Utilities Network Management System is aware of the response data in the database and displays relevant information.

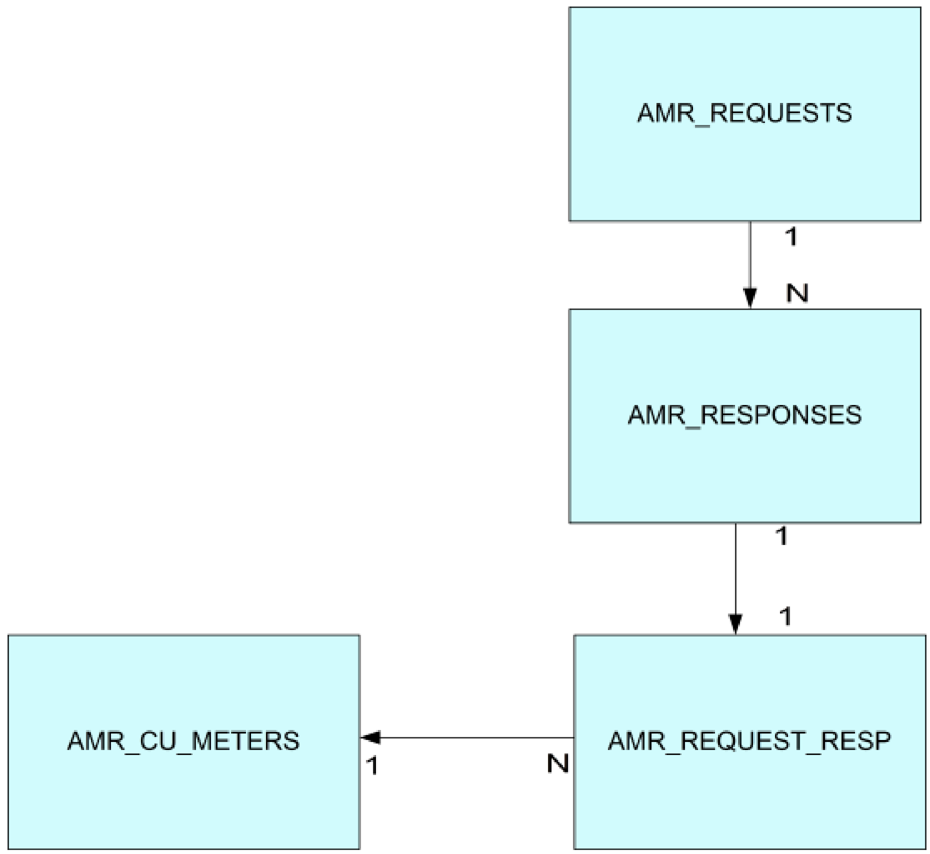

Manual Ping
Vendor Web Service
AMR Interface
OMS
Device Event Created
Operator uses Manual Ping
Poll database for new requests every X seconds
Verify AMR Meter Status for new OMS requests, generate vendor AMR request
OMS processes status, repredicts if necessary, probability and received time is updated
Send Meter Status to JMService
Poll database for new requests every X seconds

Database Schema

Oracle Utilities Network Management System MultiSpeak Adapter uses several databases tables to store meter status information received from the AMR system and pending meter ping requests.

Entity Relationship Diagram

The following diagram illustrates entity relationships for the Oracle Utilities Network Management System MultiSpeak Adapter.



AMR_REQUESTS

AMR_REQUESTS is populated by the outage prediction engine when a request for meter information is submitted. There is one row per request, and each request can involve multiple meters.

Field	DataType	Nullable	Comments
REQUEST_IDX	NUMBER	No	Instance of "amr_request". PRIMARY KEY
EVENT_CLS	NUMBER	No	Event cls for which this status was requested
EVENT_IDX	NUMBER	No	Event idx for which this status was requested.
REQUEST_TIME	DATE		Time requested

Field	DataType	Nullable	Comments
WHO_REQUESTED	VARCHA R2(32)		User name of requestor
WHO_COMPLETED	VARCHA R2(32)		User name who completed (or cancelled)
STATUS	NUMBER		Status of this AMR request. This will map to an enumerated list. 1- active 2 - explicitly completed 3 -cancelled
DEVICE_CLS	NUMBER		Device class where AMR request initiated (or Job device)
DEVICE_IDX	NUMBER		Device index where AMR request initiated (or Job device)
NCG	NUMBER		NCG of the device_cls, device_idx

AMR_RESPONSES

AMR_REQUESTS is populated by the outage prediction engine when a request for meter information is submitted. Every request in AMR_REQUESTS is represented, with one row for each meter requested..

Field	DataType	Nullable	Comments
REQUEST_IDX	NUMBER	No	Instance of "amr_request". PRIMARY KEY
METER_ID	VARCHAR2(32)	No	Meter id string. PRIMARY_KEY
RECEIVED_TIME	DATE		Time update received. Initially NULL
PROBABILITY	NUMBER		Probability on/off (0-100)
STATUS	VARCHAR2(256)		Response status string

AMR_CU_METERS

Table AMR_CU_METERS contains information about all meters known to the Oracle Utilities Network Management System..

Field	DataType	Nullable	Comments
METER_ID	VARCHAR2(14)	No	Meter identifier in Oracle Utilities Network Management System.
METER_NO	VARCHAR2(20)	Yes	Meter identifier used by the AMR system.
RESULT_TIME	DATE	Yes	Timestamp when the order was last updated.

Field	DataType	Nullable	Comments
POWER_UP_TIME	DATE	Yes	Timestamp of the latest power-up message.
LAST_GASP_TIME	DATE	Yes	Timestamp of the latest "last gasp" message.
ALT_METER_NO	VARCHAR2(256)	Yes	Alternative meter number.
AMR_ENABLED	VARCHAR2(1)	Yes	Indicator that meter is AMR-enabled.
STATUS	VARCHAR2(256)	Yes	Meter status ("ON" or "OFF"). For testing and simulation only.

AMR_REQUEST_RESP

The AMR_REQUEST_RESP table is used to keep track of the ping requests currently being processed by the interface. It is initially populated through a trigger on the AMR_RESPONSES table, with one row added for each row in AMR_RESPONSES. Once the interface has passed meter status information to the Oracle Utilities Network Management System record for the completed request is removed from this table.

This table has a large number of columns to account for the wide variety of information that AMR systems can provide. Currently most of these columns are not used. Only the most commonly used columns are listed in the table below.

Field	DataType	Nullable	Comments
AMR_REQUEST_ID	NUMBER	No	Primary key.
REQUEST_ID	NUMBER	Yes	Request id from Oracle Utilities Network Management System.
REQUEST_TIME	DATE	Yes	Timestamp when the meter ping request was initiated.
REQUEST_TYPE	VARCHAR2 (256)	Yes	Not used.
METER_NO	VARCHAR2 (256)	Yes	Meter identifier used by the AMR system.
METER_ID	NUMBER	Yes	Meter identifier in Oracle Utilities Network Management System.
STATUS	VARCHAR2 (255)	Yes	Ping request status: N – new request; P – pending request; R – response received; C – completed request.
DEV_POWER_STATUS	NUMBER	Yes	Meter status ("ON" or "OFF").
SUCCESS_STATUS	NUMBER	Yes	Not used.
REPORT_TIME	DATE	Yes	Not used.

Field	DataType	Nullable	Comments
RESULT_TIME	DATE	Yes	Timestamp of the latest meter status received from the AMR system.
ATTEMPT_COUNT	NUMBER	Yes	Number of ping requests sent to the AMR system.
PROBABILITY	NUMBER	Yes	Probability that the meter has power (0-100).

Symbols
AVLInterface.properties 5
callbackFeederTimeout 29
defaultCallbackAgent 28
IccpAdapter.prm 3
MDS_ RELATION_FIELD 45
SRS Rules Configuration 26
A
activateCrew 76
Affected Customers 5
Aggregation 13
Alternate Names 24
amqscnxc 11
amqsgetc 11, 12
amqsputc 11
AMR Business Processes 9
AMR_CU_METERS 14
AMRInterface.properties 3
AMR_REQUEST_RESP 15
AMR_REQUESTS 13
AMR_RESPONSES 14
ANALOG_MEASUREMENTS 14
Analog_Measurements Table 13
ArcGIS 9.x Adapter 1
ArcGIS Data Maintenance 8
ArcGIS Versioning 8
arriveCrew 74
assignCrew 75
availableCrew 76
B
Base Path 35
Batch Execution 13
Batch Mode 20
C
Callback Requests 6
Callback Responses 7
callbackInterfaceEnabled 27
Callbacks application data flows 2
cancelTimeout 118
CES_GET 34
CES_PUT 34
Change Detect Log File 49
Change Detection 44, 26
classTable 102
clearIncidents 90
ClosestServiceAddress 36
Code Mapping Tables 46

completeEvent 92
completeOrder 120
Configuration Documents 37
confirmDeviceOutage 96
confirmServiceOutage 96
Controls Table 14
createCrew 70
createOrder 118
createPseudoRelation 100
createPseudoRelationFromConfirmServiceOutage 100
Crew Message Tags 31
Crew Outage Status Changes 17
crewActive 77
Customer Message Tags 29
Customer Outage History 15
customerPhoneParentheses 28
D
damageSetAddress 81
damageSetComment1 86
damageSetCrewId 80
damageSetFeederName 82
damageSetGrid 83
damageSetLoadAffected 85
damageSetLocation 84
damageSetNcg 82
damageSetOption1 86
damageSetPhase 85
damageSetReportTime 81
damageSetSection 84
damageSetText1 86
damageSetText2 87
damageSetTransformer 83
damageSetType 87
damageSetZoneName 83
Data Flows 14
data flows 1
Database Table Schema 32
Dataless_rwo_joins 7
decodeDateTime 110
Defaults Table 9
deleteRelation 102
Digital_Measurements Table 12
DIGITAL_MEASUREMENTS 15
dispatchCrew 73
DML Examples 49
DML Files 16
DML Function Calls 69

DML Reference 56
E
ert 94
ESRI ArcGIS 9.x Adapter 1
Event Object Fields 123
Event Status 5
eventIsActive 95
ExcludeComponent Table 15
ExcludeFeature Table 14
External Objects 27
Extract Log File 48
F
findCrewByExternalKey 71
findCrewByExtKeySubStr 72
findCrewById 71
findCrewByIdSubStr 72
findEventObject 121
findOrCreateDamage 80
findOrder 119
findRelation 99
Flags 23
formatClues 116
formatDateTime 110
formatDTNow 111
Freeze Naming Conventions 39
Freeze Versions 38
function
    createCrew 70
G
G/Electric Adapte 1
Generic IVR Adapter 1
Generic SCADA Adapter 1
Generic WebSphere MQ Adapter 1
Generic WebSphere MQ Mobile Adapter 1
Geometric Information 4
getCaseNotesForEvent 91
getClassDesc 103
getGuid 104
GIS Data Requirements 24
H
haveEventOrder 120
haveOrder 119
High Level Messages 9
High Priority Category Table 48
highPriTCCategoriesFromClues 114
I
ICCP Adapter 1

Incident Object Fields 127
Incidents table 25
Information Model 20
Input Documents 12
Input Element Arrays 28
Installing ArcGIS 23
interfaceDown 105
interfaceUp 105
Intergraph G/Electric Adapter 1
isCls 103
isDigits 108
isIn 112
isSet 106
IVR Adapter 1
IVR Adapter Installation 9
IVR data flows 1
IVRAdapter command line options 10
J
JBoss Application Server 2
L
length 107
Listener 8
listener 10
loadMapConfigTable 78
loadMapTable 78
loadTroubleCodes 115
lockForEventCreation 97
logDebug 89
logFatalError 89
logLocalError 88
M
Map Tiles 6
mapTableCode 79
mapTableStr 79
MDS_ORDER 41
MDS_ORDER_FIELD 42
MDS_RELATION 44
MDS_RELATION_EVENT 45
Message Tags 29
Model Preprocessor File 47
Modeling Stations and Substations 14
MQ Client 10
MultiSpeak Adapter 1
N
NCG 12
Network Component Group 12
NMSDeviceID 10

Non_joined_connected_objects 8
O
OMSA Object ID 7
OMSA_additional_info 8
OMSA_associated_objects 7
OMSA_component 9
OMSA_extract_world 8
OMSA_internal_associated_objects 7
OMSA_proto_record 8
Oracle WebLogic Server 2
Order Tables 40
Outage Detection 9
Outage Status 15
Output Documents 12
Output Documents 12
P
Partition Listbox 45
PartitionGroupFeature Table 15
Partition_Reference Table 11
Partitions 4
patialAttributeQueries 35
pause 112
PDO Verification 10
Permanent Order Object Fields 129
Permanent Relationship Object Fields 129
phaseInt 116
phaseStr 116
picklistCompLog 98
picklistInfoUpdTr 99
pr_affected_customers 53
pr_customer_event_details 58
Prime Location 7
printEvntCrew 122
pr_trouble_callback_requests 56
pr_trouble_callback_responses 57
pr_trouble_calls 47
pr_trouble_status 51
Pseudo Relationships 40
PSO Verification 9
Q
Q manager listener 10
Quality Codes 8
Quality Rules Table 11
Queue Manager Listener Service 8
Queue Manager 7
Queue Specification 31
Queues 32

R
RDBMS configuration 3
readGenericField 93
readIncidents 90
releaseCrews 77
removeDelim 109
requestRowAction 94
requestRowActionAll 95
Restoration Verification 11
restoreOutage 97
Root Element 34
RS Message Type Table 47
RTI_STATUS 12
RTI_SYN Database 11
Run Time Errors 49
S
saveDamageDetails 88
SCADA Adapter 1
SCADA Adapter Data Flows 9
SCADA Measurement Entry 9
SCADA_ID Database 11
selectValue 113
Server Connection Channel 8
setAlarm 104
setCaseNoteInfo 91
setDocValue 121
setEventInfo 92
setGenericField 93
setTimeout 117
SmallWorld GIS Adapter 1
sortIncidents 114
Spatial (Landbase) Partition List 14
sql 105
SQL Query 16
SQL Transactions 16
SRS Output 21
Stop Statement 36
Stored Procedure Parameters 47
streetXsectionOffset 29
stringbefore 108
stringInString 109
substring 107
Synchronization Process 12
System Partitions 4
T
time 111
triggerOutputDoc 113

triggerRelationChanged 101
Triggers 31
Trouble Call Mapping Configuration 18
Trouble Call Message Tags 29
Trouble Call Performance 30
Trouble Calls 4
TROUBLE_AFFECTED_CUSTOMERS 41
TROUBLE_CALLBACKS 42
troubleCallCreate 16
TROUBLE_CALLS Table Schema 32
TROUBLE_STATUS Table Schema 39
U
unassignCrew 75
updateCrew 73
useExternalCause 27
V
Visibilities 6
V_SPL_FIDFNO View 16
W
WebSphere MQ Adapter 1
WebSphere MQ Mobile Adapter 1
WebSphere MQ Server 9
Windows Resources 51
Work Order Time Stamp File 49
X
xml 122