## Contents

### Preface

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audience</td>
<td>xiii</td>
</tr>
<tr>
<td>Documentation Accessibility</td>
<td>xiii</td>
</tr>
</tbody>
</table>

### 1 Installing Design Studio for Network Integrity

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing Studio for Network Integrity</td>
<td>1-1</td>
</tr>
<tr>
<td>Installing the Network Integrity Plug-Ins</td>
<td>1-1</td>
</tr>
<tr>
<td>Studio Perspectives</td>
<td>1-2</td>
</tr>
<tr>
<td>Studio Views</td>
<td>1-2</td>
</tr>
<tr>
<td>Studio Design Perspective Views</td>
<td>1-2</td>
</tr>
<tr>
<td>Java Perspective Views</td>
<td>1-3</td>
</tr>
</tbody>
</table>

### 2 Setting Preferences in Network Integrity

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Design Studio to Set Preferences</td>
<td>2-1</td>
</tr>
<tr>
<td>Configuring Network Integrity Preference Settings</td>
<td>2-1</td>
</tr>
<tr>
<td>Configuring Data Dictionary Preference Settings</td>
<td>2-2</td>
</tr>
</tbody>
</table>

### 3 Working With Cartridges

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Cartridges</td>
<td>3-1</td>
</tr>
<tr>
<td>About Cartridges and Design Studio</td>
<td>3-2</td>
</tr>
<tr>
<td>About the Cartridge Architecture</td>
<td>3-2</td>
</tr>
<tr>
<td>Creating a Cartridge</td>
<td>3-3</td>
</tr>
<tr>
<td>Working with the Network Integrity Cartridge Editor</td>
<td>3-4</td>
</tr>
<tr>
<td>Properties Tab</td>
<td>3-5</td>
</tr>
<tr>
<td>Copyright Tab</td>
<td>3-5</td>
</tr>
<tr>
<td>Model Variables Tab</td>
<td>3-5</td>
</tr>
<tr>
<td>Cartridge Management Variables Tab</td>
<td>3-5</td>
</tr>
<tr>
<td>UI Hints Tab</td>
<td>3-5</td>
</tr>
</tbody>
</table>

### 4 Working with Actions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions and Processors</td>
<td>4-1</td>
</tr>
<tr>
<td>About the Processors Tab</td>
<td>4-2</td>
</tr>
<tr>
<td>Adding an Existing Action</td>
<td>4-3</td>
</tr>
<tr>
<td>Generated Action MDB and Controller</td>
<td>4-5</td>
</tr>
</tbody>
</table>
5 Working with Processors

Creating Processors ............................................................................................................... 5-2
Configuring Processors ........................................................................................................ 5-2
About Context Parameters ................................................................................................. 5-2
About Properties and Property Groups ......................................................................... 5-2
About Generated Code .......................................................................................................... 5-3
  About the Location for Generated Code ...................................................................... 5-3
  About the Processor Interface ....................................................................................... 5-4
  About the PropertyGroup and Properties Classes ...................................................... 5-4
About Processor Implementation ...................................................................................... 5-5
About the Processor Finalizer .......................................................................................... 5-5
  ProcessorFinalizer .......................................................................................................... 5-5
  About Memory Considerations ...................................................................................... 5-6

6 Working with Discovery Actions and Processors

About Supported Processors ........................................................................................................ 6-1
Creating a Discovery Action ........................................................................................................ 6-1
Configuring a Discovery Action .............................................................................................. 6-2
  About Address Handlers .................................................................................................... 6-2
  About Result Categories ...................................................................................................... 6-2
About the Discovery Action in the Network Integrity User Interface .................................. 6-3
About UI Parameters ................................................................................................................. 6-4
Creating Discovery Processors .............................................................................................. 6-5
  Configuring a Discovery Processor ................................................................................... 6-5
  Implementing a Discovery Processor .............................................................................. 6-5
  Implementation Code Sample ............................................................................................. 6-6

7 Working with SNMP Processors

Creating an SNMP Processor .................................................................................................... 7-1
  Configuring the SNMP Processor ...................................................................................... 7-2
  About the Generated Implementation and XML Beans .................................................. 7-2
  Supporting New MIBs ......................................................................................................... 7-3

8 Working with Import Actions and Processors

Creating Import Actions .......................................................................................................... 8-1
Configuring Import Actions ..................................................................................................... 8-2
  Details Page ....................................................................................................................... 8-2
  Processors Page .................................................................................................................. 8-2
  UI Parameters Page ............................................................................................................. 8-3
  Model Page .......................................................................................................................... 8-3
  Conditions Page ................................................................................................................... 8-3
12 Working with Discrepancy Resolution Actions and Processors

Creating Discrepancy Resolution Actions ................................................................. 12-2
Discrepancy Resolution Action Editor ...................................................................... 12-2
Details Page .................................................................................................................. 12-2
Processors Page ............................................................................................................ 12-2
Model Page .................................................................................................................... 12-2
Result Source Page ....................................................................................................... 12-3
Conditions Page ............................................................................................................ 12-3
About the Resolution Action Label .............................................................................. 12-3
About Result Sources .................................................................................................... 12-4
Generated Action MDB and Controller ....................................................................... 12-5
About Discrepancy Resolution Processors ................................................................. 12-5
Creating a Discrepancy Resolution Processor ............................................................ 12-5
Implementing a Discrepancy Resolution Processor ...................................................... 12-5
About the Implementation Interface ............................................................................. 12-5
About the Input Parameters of the Invoke() Method .................................................. 12-6
Return Type of Invoke() ............................................................................................. 12-6
About the General Flow of the Discrepancy Resolution Processor ......................... 12-6
Fetching Discrepancies ................................................................................................. 12-7
Grouping Discrepancies .............................................................................................. 12-7
Handling Discrepancies .............................................................................................. 12-7
Reporting the Resolution Result .................................................................................. 12-8
Handling Discrepancies Asynchronously ..................................................................... 12-8

13 Working with Address Handlers

Creating Address Handlers .......................................................................................... 13-2
Specifying the Address Handler Implementation ........................................................ 13-2
About the Address Handler Interface .......................................................................... 13-2
14 **Working with Result Categories**

About Result Categories ................................................................. 14-1
About Result Categories in Network Integrity................................. 14-2

15 **Model Extension Using Specifications**

Overview .......................................................................................... 15-1
Working with Specifications ............................................................... 15-2
Data Dictionary Editor ........................................................................ 15-2
Model Collections ............................................................................. 15-3
Create Model Collection................................................................. 15-3
Adding a Specification to a Model Collection ................................. 15-3
Removing a Specification from a Model Collection......................... 15-3
Action Model Tab ........................................................................... 15-3
Adding a Model Collection to a Model ............................................. 15-4
Removing a Model Collection from an Model ................................. 15-4
Cartridge Editor UI Hints Tab .......................................................... 15-4
Specification Usage Examples ......................................................... 15-4
Create and Use Specification Example ............................................. 15-4
Use an Existing Specification Example ............................................ 15-5
Modify Specification Usage Example .............................................. 15-5
Defining Enumerated Values for Specifications Example ............... 15-5
Stop Using a Specification Usage Example ....................................... 15-5
Specification Helpers ...................................................................... 15-6

16 **POMS SDK**

About POMS ..................................................................................... 16-1
Working with POMS Entities ............................................................ 16-2
Working with POMS Relationships .................................................. 16-2
One-to-one Relationships ............................................................... 16-2
One-to-Many or Many-to-Many Relationships ................................. 16-2
Ordered and Unordered Relationships .......................................... 16-3
Bi-directional Relationships ........................................................... 16-3
Relationship Entities ...................................................................... 16-3
Working with Specifications and Characteristics ........................... 16-3
Working with the POMS Finder ......................................................... 16-4
Find by Entity .................................................................................. 16-4
Find by JPQL .................................................................................... 16-5
Find with Paged Results .................................................................. 16-5
POMS SDK Interfaces ..................................................................... 16-6
About Persist Results ...................................................................... 16-7

17 **Working with UI Parameters**

Associating Data Dictionary Item Types with Attributes .................. 17-1
# Table of Contents

**Chapter 17: Working with UI Parameters**
- Associating UI Attributes with Data Dictionary Structures .................................................... 17-1
- Associating UI Attributes with Data Dictionary Child Elements .............................................. 17-2
- Displaying UI Parameters in the Network Integrity User Interface ........................................... 17-2

**Working with UI Parameters** ................................................. 17-4
- About the UI Parameters Tab .................................................................................................... 17-4
- About the Data Dictionary ....................................................................................................... 17-4
- About the UI Hints Tab ......................................................................................................... 17-5

**Using UI Parameters in Actions** ............................................ 17-6
- About the Action Editor UI Parameters Tab ............................................................................ 17-6
- About the Data Dictionary Editor ............................................................................................. 17-7
- About the Cartridge Editor UI Hints Tab ................................................................................... 17-7

**Working with UI Parameter Examples** .................................... 17-7
- Creating and Using UI Parameters ....................................................................................... 17-7
- Using Existing UI Parameters ............................................................................................... 17-8
- Editing UI Parameters ........................................................................................................... 17-9
- Defining a UI Parameters Enumerated Values Example ....................................................... 17-9
- Deleting a UI Parameter ........................................................................................................ 17-10

**Chapter 18: Setting Conditions in Network Integrity**
- About Conditions ................................................................................................................. 18-1
- About Generated Classes and the Implementation Class ......................................................... 18-1
- Applying Conditions to Processors ........................................................................................... 18-2
- Importing Actions with Conditions .......................................................................................... 18-2
- Creating Condition Examples ................................................................................................... 18-3

**Chapter 19: Extensibility SDK**
- About Extensibility Scenarios ................................................................................................. 19-1
- Extending MIB II SNMP Discovery for Updated Vendor and Interface Type ........................ 19-2
  - Creating a New Project, Discovery Action, and Discovery Processor ................................. 19-2
- Extending the Existing Cartridge to Discover and Reconcile New Characteristics .............. 19-4
  - Creating Project, Discovery Action with Processors, and Adding Characteristics and Specifications 19-5
  - Creating a Discrepancy Detection Action with Processors .................................................. 19-8
  - Creating a Discrepancy Resolution Action with Processors ................................................ 19-8
  - Creating and Import Action with Processors (Optional) ...................................................... 19-9
- Extending the MIB II SNMP Discovery to Change Interface Name Value ............................ 19-10
  - Creating a Discovery Action with Processor ....................................................................... 19-11
- Multiple Vendor SNMP Discovery ........................................................................................... 19-13
  - Create a Discovery Action with Processors and Conditions ............................................. 19-14
- Multiple Protocol Discoveries ................................................................................................ 19-17
  - Create a Discovery Action with Processors and Conditions ............................................. 19-17

**Chapter 20: Building and Packaging Cartridges**
- Building Cartridges Automatically ....................................................................................... 20-1
21 Deploying and Undeploying Cartridges

Creating a Design Studio Environment Project ......................................................... 21-1
  Creating a Studio Environment ............................................................................. 21-2
Switching to the Studio Environment Perspective ..................................................... 21-4
  Connecting to the Studio Environment .................................................................. 21-5
Deploying a Cartridge ............................................................................................... 21-6
  Redeploying a Cartridge ....................................................................................... 21-7
Undeploying a Cartridge ............................................................................................ 21-7

22 Debugging and Testing Cartridges

Starting the WebLogic Server in Test Mode ............................................................... 22-1
Configuring Remote Debugger in Design Studio ...................................................... 22-2

23 Sealing and Unsealing Cartridges

24 Exporting and Importing Cartridges

Exporting a Cartridge With Source Code ................................................................. 24-1
  Modifying the Classpath ...................................................................................... 24-2
Exporting a Cartridge Without Source Code .......................................................... 24-2
  Exporting the Cartridge ...................................................................................... 24-5
Importing a Cartridge .............................................................................................. 24-5

25 Design Studio Source Control

Working with Source Control .................................................................................. 25-1

26 Web Service API

Web Service Overview ............................................................................................ 26-1
Security .................................................................................................................... 26-1
Model Based .......................................................................................................... 26-1
Concurrency With UI and other Web Service Clients .............................................. 26-2
All Operations ......................................................................................................... 26-2
Create, Read, Update, Delete Pattern ................................................................. 26-9
Create Operation .................................................................................................. 26-9
  Entity Type Support .......................................................................................... 26-10
Get Operation ....................................................................................................... 26-10
  Entity Type Support .......................................................................................... 26-10
Get All Operation ................................................................................................. 26-12
Delete Operation .................................................................................................. 26-13
  Entity Type Support .......................................................................................... 26-13
26-14
Entity Type Support................................................................. 26-15

Find Operation ........................................................................................................................................ 26-15
From and To Range ................................................................................................................................... 26-16
Ascending & Descending ....................................................................................................................... 26-16
Attribute Criteria ....................................................................................................................................... 26-16
Multiple Attribute Criteria .................................................................................................................... 26-17
Extended Attribute Criteria .................................................................................................................. 26-18
Criteria Operators ..................................................................................................................................... 26-18
DisConfig ............................................................................................................................................... 26-18
DisScanRun ........................................................................................................................................... 26-19
DisDiscrepancy ....................................................................................................................................... 26-20
Between/Not Between Operator ............................................................................................................. 26-21
Date Criteria ............................................................................................................................................ 26-21
Conjunction Criteria .............................................................................................................................. 26-21
Find Response .......................................................................................................................................... 26-23

Special Function Operations .................................................................................................................. 26-23
Start Scan .................................................................................................................................................. 26-23
Stop Scan .................................................................................................................................................. 26-23
Get Latest Scan Status ............................................................................................................................. 26-24
Submit Discrepancies For Resolution Processing ................................................................................ 26-25

High Level Scenarios ................................................................................................................................ 26-26
Creating a Scan .......................................................................................................................................... 26-26
Starting, Stopping, and Monitoring a Scan .............................................................................................. 26-27
Retrieving Scan Results/Data .................................................................................................................. 26-27
Working with Discrepancies ................................................................................................................... 26-28

Network Integrity Web Service Samples .................................................................................................. 26-28
Contents of the Network Integrity Web Service Samples ZIP File ......................................................... 26-28
Sample Java Client .................................................................................................................................... 26-29
Sample Soap UI Project .......................................................................................................................... 26-29
Submitting Request to the Server ........................................................................................................... 26-30
Specifying User Name and Password in Request .................................................................................. 26-30

27 Working with Scan Run Complete Notification

About Clients for Monitoring Scan Run Complete Notification Messages ........................................... 27-1

28 Tips and Tricks

Processors ..................................................................................................................................................... 28-1
Creating Processors ................................................................................................................................... 28-1
Specifying Context Parameters before Creating Implementation Class ............................................. 28-1

Working in Design Studio ........................................................................................................................ 28-1
About Package Explorer View and Navigator View ................................................................................... 28-2
About Error Log View ............................................................................................................................... 28-2
Opening with the Wrong Editor .............................................................................................................. 28-2

Compile or Build Problems ...................................................................................................................... 28-2
About Java Errors in the Generated Controller Class ............................................................................ 28-2
Renaming or Deleting Actions and Processors ....................................................................................... 28-3
29 Network Integrity Plug-in Validation Error Messages

Error Message Classifications and Conditions ................................................................. 29-1
Design Studio Logging ......................................................................................................... 29-8

30 JCA Resource Adapters

About Resource Adapters ................................................................................................. 30-1
Understanding JCA Resource Adapter Connectivity Options ............................................ 30-2
Understanding JCA Resource Adapters with Network Integrity ........................................ 30-2
About Productized SNMP JCA Resource Adapter ............................................................. 30-3
Installing the SNMP JCA Resource Adapter ................................................................... 30-3
Extending the SNMP JCA Resource Adapter .................................................................... 30-3
Record and Playback Mode ............................................................................................... 30-4
Invoking the SNMP JCA Resource Adapter in a Network Integrity Cartridge .................... 30-5
About Third Party or Customized JCA Resource Adapters .................................................. 30-5
Building a JCA Resource Adapter in Weblogic ................................................................. 30-5
Invoking a Third Party or Customized JCA Resource Adapter ............................................ 30-6

31 Network Integrity Reports Extensibility

About BI Publisher .............................................................................................................. 31-1
About BI Publisher Desktop ............................................................................................... 31-1
Reports Provided with Network Integrity ........................................................................... 31-1
Scan History Report ........................................................................................................... 31-2
Discovery Scan Summary Report ....................................................................................... 31-2
Device Discrepancy Detection Summary Report ............................................................... 31-2
Device Discrepancy Detection Detail Report ..................................................................... 31-2
Discrepancy Corrective Action Report ............................................................................... 31-3
Network Integrity Report Building Blocks ........................................................................ 31-3
RTF Templates .................................................................................................................... 31-3
Report Definition Files ...................................................................................................... 31-4
Data Source Query Tools ................................................................................................. 31-4
Parameters ........................................................................................................................... 31-4
List of Values ....................................................................................................................... 31-4
Rendering Options ............................................................................................................ 31-4
Report Properties ................................................................................................................ 31-4
Developing BI Publisher Reports ........................................................................................ 31-5
Report Requirements ......................................................................................................... 31-5
Configuring a Data Source Tutorial .................................................................................. 31-5
Creating a Report Tutorial ................................................................................................. 31-6
Building an RTF Template Tutorial .................................................................................. 31-7
Using Microsoft Word Native Features and XSL ............................................................... 31-9
Using BI Publisher Features

31-9
  Scheduling Reports ................................................................. 31-9
    Enabling BI Publisher Scheduler Tutorial .......................... 31-9
    Using BI Publisher to Schedule Reports Tutorial ............ 31-10
  Localizing Reports ............................................................... 31-10
    Localized Template Option .............................................. 31-10
    XLIFF File Option ............................................................ 31-11

Installing and Integrating BI Publisher with Network Integrity

31-12
  Installing BI Publisher in Standalone Mode .................. 31-12
  Deploying BI Publisher ....................................................... 31-12
  Integrating BI Publisher with Network Integrity using WebLogic Enterprise Manager .... 31-12
  Integrating BI Publisher with Network Integrity using JConsole .................. 31-13
  Integrating BI Publisher with the Network Integrity Installer ........... 31-14

Uploading and Exporting Reports

31-14
  Exporting a Report .............................................................. 31-14
  Uploading a Report ............................................................ 31-15

32 Network Integrity SOA Extensibility

About SOA Extensibility ............................................................. 32-1
  Purpose of Documentation .................................................. 32-1

Extensibility Tasks .................................................................. 32-1
  Extensibility Tasks .............................................................. 32-2
  Installing Oracle Weblogic Server .................................... 32-2
  Installing Oracle JDeveloper ............................................. 32-3
  Installing Oracle Application Runtime ............................... 32-4
  Installing Oracle SOA Suite ................................................ 32-4
  Creating SOA Metadata Service Schemas ......................... 32-5
  Updating JDeveloper for Latest SOA Composite Editor ........ 32-6
  Creating WebLogic Domain with SOA Products ............... 32-7
  Creating and Updating Sample SOA Application Using Network Integrity Web Service .. 32-8
  Starting and Stopping SOA Servers ................................... 32-9
  Building and Deploying the SOA Application .................... 32-10
  Testing Sample SOA application ........................................ 32-11
    Testing Network Integrity SOA Application Using EM .......... 32-11
    Testing Network Integrity SOA Application Using soa-infra .... 32-11
    Testing Network Integrity SOA Application Using SOAP UI Tool ........................................ 32-11

33 Network Integrity Localization Pack

Localization Pack Overview .................................................... 33-1
  Localization Summary ......................................................... 33-1
  Localization Pack Summary ............................................... 33-1
  Expected Outcomes ............................................................ 33-1

Building the Localization Pack ................................................. 33-1

Testing the Browser Local ........................................................ 33-4
Preface

This guide explains how to extend Oracle Communications Network Integrity through standard Java practices using Oracle Communications Design Studio, which is an Eclipse-based integrated development environment. This guide includes references to both applications, and often directs the reader to see the Studio online Help and the Network Integrity online Help for instructions on how to perform specific tasks.

This guide should be read after reading Oracle Communications Network Integrity Concepts, because this guide assumes that the reader has a conceptual understanding of Network Integrity. This guide should be read from start to finish because the information presented in a chapter often builds upon information presented in a preceding chapter.

This guide refers to Oracle Communications Information Model Reference and Network Integrity Information Model Reference, which are part of the 7.2.2 Network Integrity documentation set.

This guide includes examples of typical development code used in given situations. The guidelines and examples may not be applicable in every situation.

Audience

This guide is intended for developers who implement code to extend Network Integrity. The developers should have a good working knowledge of XML and Java development and, in particular, JDO, standard Java practices, and J2EE principles.

You should read Oracle Communications Network Integrity Concepts before reading this guide.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.
This chapter provides information on Design Studio, an Eclipse-based integration development environment. Studio comes with plug-ins specific to Network Integrity that enable you to extend Network Integrity.

Installing Studio for Network Integrity

Use Design Studio to extend several Oracle products, one of which is Network Integrity. Plug-ins are available for the different Oracle products, and each plug-in provides JAR or IAR (Integrity ARtifact) files that are unique to the product. Because Studio is used by several different Oracle products, the information on how to install Studio is centrally located.

For directions on how to install Studio, which includes the installation of Eclipse, see Design Studio Installation Guide.

Installing the Network Integrity Plug-Ins

For directions on how to install the Network Integrity plug-ins, see Design Studio Installation Guide. The section about installing Studio features into Eclipse describes how to install all available Oracle Communications plug-ins.

Ensure the following features are configured in the site.xml file for Network Integrity:

- Design Studio Platform
- Design Studio for Network Integrity
- Design Studio for POMS

The following is an example of a site.xml file that is configured with the required Network Integrity features:

```xml
<?xml version="1.0" encoding="UTF-8"?>

<site>
  <description url="<UPDATE_SITE>">
  Oracle Communications Design Studio update site.
  </description>
  <feature url ="features/oracle.communications.sce.core.
  feature_version_build_number"
    id ="oracle.communications.sce.core.feature" version="version_build_number">
    <category name="Oracle Communications Design Studio"/>
  </feature>
  <feature url ="features/oracle.communications.sce.integrity.
  feature_version_build_number.jar"
    id="oracle.communications.sce.integrity.feature" version="version_build_number">
```
Studio Perspectives

Perspectives define the Workbench layout and provide functionality for working with different types of resources. Several perspectives are available within Studio. The Studio Design perspective and the Java perspective are commonly used when extending Network Integrity, and the use of these perspectives is mentioned throughout this guide.

For instructions on how to open a perspective in Studio, see the Studio online Help.

Studio Views

Within a given perspective, views further define the Workbench layout and provide different presentations of resources. Several views are available within Studio, and the available views are dependent upon the perspective.

For instructions on how to open a view in Studio, see the Studio online Help.

Studio Design Perspective Views

Figure 1–1 shows the available views when in the Studio Design perspective. When extending Network Integrity, commonly used views are Cartridge and Package Explorer.

Figure 1–1  Studio Design Perspective Views

- Blueprint
- Cartridge
- Dictionary
- Outline
- Overview
- Package Explorer  Alt+Shift+O, P
- Problems
- Properties
- Relation
- Relation Graph
- Solution
- Other…  Alt+Shift+Q, O
Java Perspective Views

Figure 1–2 shows the available views when in the Java perspective. When extending Network Integrity, commonly used views are Navigator, Package Explorer, and Error Log.

Figure 1–2 Java Perspective Views

<table>
<thead>
<tr>
<th>View</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art</td>
<td>Alt+Shift+Q, C</td>
</tr>
<tr>
<td>Console</td>
<td>Alt+Shift+Q, C</td>
</tr>
<tr>
<td>Declaration</td>
<td>Alt+Shift+Q, D</td>
</tr>
<tr>
<td>Error Log</td>
<td>Alt+Shift+Q, L</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Alt+Shift+Q, T</td>
</tr>
<tr>
<td>Javadoc</td>
<td>Alt+Shift+Q, J</td>
</tr>
<tr>
<td>Navigator</td>
<td></td>
</tr>
<tr>
<td>Outline</td>
<td></td>
</tr>
<tr>
<td>Package Explorer</td>
<td>Alt+Shift+Q, P</td>
</tr>
<tr>
<td>Problems</td>
<td></td>
</tr>
<tr>
<td>Progress</td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td>Alt+Shift+Q, S</td>
</tr>
<tr>
<td>Task List</td>
<td></td>
</tr>
<tr>
<td>Tasks</td>
<td></td>
</tr>
<tr>
<td>Other…</td>
<td>Alt+Shift+Q, Q</td>
</tr>
</tbody>
</table>
Oracle Design Studio is used to set preferences for Oracle Communications Network Integrity and for the data dictionary.

Using Design Studio to Set Preferences

To configure Network Integrity, see:

- Configuring Network Integrity Preference Settings

To configure the data dictionary, see:

- Configuring Data Dictionary Preference Settings

Configuring Network Integrity Preference Settings

To configure Network Integrity preferences:

1. From the Design Studio Menu, select Window, and then Preferences.
   The Preferences dialog box appears.
2. In the left-column menu tree, expand the Oracle Design Studio folder.
3. Click Network Integrity.
4. In the Default Cartridge Package field, specify the default cartridge package in this text field.

   By default, the cartridge package is com.organization.integrity, which is automatically filled in the Default Package field for new Network Integrity cartridges created in Design Studio.

   Changing this default package does not affect the existing cartridges.
5. In the MIB Directory field, specify the directory that contains all of the MIB files that Studio uses to create an SNMP processor.
   Enter the MIB directory name or browse to the location of the directory.
6. Click OK to set the MIB directory. If the MIB directory is not properly set, the SNMP processor cannot be created in Design Studio.
7. Click Apply.
8. Click OK to restart the Design Studio workbench.
Configuring Data Dictionary Preference Settings

You configure data dictionary preference settings to specify the horizontal depth to which any data dictionary tree can expand.

To configure data dictionary preferences:

1. From the Design Studio Menu, select **Window**, and then **Preferences**.
   
   The Preferences dialog box appears.

2. In the left-column menu tree, expand the Oracle Design Studio folder.

3. Click **Data Dictionary**.

4. In the **Expansion level** field, specify the horizontal depth to which data dictionaries can expand. For Network Integrity Design Studio, the data dictionary recursion level should be set to 4 to import the Oracle Communications Information Model cartridge.

   See Design Studio Help, Defining Preferences, for more information.

5. Click **Apply**.

6. Click **OK** to restart the Design Studio workbench.
This chapter provides an overview of Oracle Communications Network Integrity cartridge concepts and procedures.

About Cartridges

Network Integrity cartridges are packaged extensions to the core application. They represent the necessary components needed for the following:

- Discovering network elements, either from a Network Management System (NMS) or through direct contact with the Network Element (NE)
- Importing network elements from an inventory system
- Assimilating network data using business logic
- Detecting discrepancies between the network and the inventory system
- Resolving discrepancies, either within the network, or in the inventory system

Cartridges provide the ability to support new functionality as business cases arise, such as:

- New protocols; for example: Command Line Interface (CLI), Transport Layer Security (TLS), and so on
- New standards; for example: a new RFC
- New vendor devices; for example: Juniper, Huawei, and so on
- New operational or business support systems

Cartridges are made up of one or more of the following components:

- **Actions**: Network Integrity entities that represent a broad action; for example: Discovery MIB II SNMP. Actions are made up of one or more processors to accomplish a task. See "Working with Actions" for more information.

- **Processors**: Network Integrity entities that represent a specific atomic sub-function within an action. See "Working with Processors" for more information.

- **Model collections**.
  See "Model Extension Using Specifications" for more information.

- **Address Handlers**.
  See "Working with Address Handlers" for more information.
About Cartridges and Design Studio

Cartridges contain Network Integrity entities, such as Actions and Processors. Design Studio is the tool used by customers, systems integrators, and third-party vendors to develop cartridges that meet their business needs.

When extending Network Integrity, you can create one or many cartridges, depending on how you choose to organize the extensions.

Everything you create in Design Studio resides in a cartridge. The name you choose for the cartridge becomes the name of the IAR (Integrity Archive) file, and everything you create within that cartridge is automatically placed in the IAR file.

About the Cartridge Architecture

Figure 3–1 shows the cartridge architecture.

In the context of a cartridge, address handlers and actions cannot coexist. Therefore, address handlers must be defined in their own cartridge. This allows for a clear segregation of responsibility. So, for example, cartridge developers create a cartridge called AddressHandlers where different address handler types exist (for example: IP...
Address, URL, and so on) and simply reference those from within their discovery and import cartridges.

Cartridges can also reuse actions from other cartridges to extend behavior. For example, a Juniper-specific SNMP cartridge (that is, containing Juniper MIBs) could extend a generic SNMP Cartridge (MIB II only).

After all components are defined, cartridges are packaged into an IAR (Integrity Archive) file and can be deployed to a running Network Integrity system using the Cartridge Deployer. Use the Cartridge Deployer to deploy cartridges to production environments. Alternatively, cartridges can be deployed through Design Studio on non-production environments.

See "Building and Packaging Cartridges" and "Deploying and Undeploying Cartridges" for more information.

After a cartridge is deployed, it is available to Network Integrity.

To determine whether a cartridge is deployed in Network Integrity:

1. From the Network Integrity main menu, click Help, and then select About. The Network Integrity components dialog appears.

   Figure 3–2 shows the Network Integrity components dialog.

   **Figure 3–2 Network Integrity Components Dialog**

   ![Network Integrity Components Dialog](image)

2. Select the Components tab.

   The Network Integrity product version is displayed with the versions of all cartridges deployed in Network Integrity.

**Creating a Cartridge**

To create a Network Integrity Cartridge using Design Studio:

1. From within Design Studio, switch to the Design perspective.
2. Right-click in a blank area inside the Cartridge View pane, and select New and then select Integrity Project. The New Studio Product Cartridge Project dialog box appears.

3. In the Project name field, enter a project name.

4. Click Finish to create the cartridge.

You can now create actions or address handlers.

Figure 3–3 shows an example of a cartridge as it appears in the Studio Design Cartridge Editor.

**Figure 3–3  Network Integrity Sample Cartridge**

---

**Working with the Network Integrity Cartridge Editor**

This section details the contents of the Network Integrity Cartridge Editor, as shown in Figure 3–3.

The Network Integrity Cartridge Editor consists of the following tabs:

- Properties Tab
- Copyright Tab
- Model Variables Tab
- Cartridge Management Variables Tab
- UI Hints Tab
Properties Tab

The Properties tab contains general information about the cartridge, such as a description, versioning, state (that is, sealed or unsealed) and a default package to use when auto-generating code.

Copyright Tab

The Copyright tab contains text boxes where copyright and license information can be added.

Model Variables Tab

The Model Variables tab contains a list of property value pairs.

Note: Network Integrity does not currently use Model Variables.

Cartridge Management Variables Tab

The Cartridge Management Variables tab contains a list of properties used when deploying or undeploying cartridges from Design Studio. These include:

- `wladmin.host.name` — the host name of the server where the WebLogic Server resides
- `wladmin.host.port` — the WebLogic Server port number
- `wladmin.server.name` — the WebLogic Server Admin Server name (for example, `AdminServer`)

UI Hints Tab

The UI Hints tab contains the set of UI Hints associated to this cartridge. See "Working with UI Parameters" for more information.
Actions are Network Integrity entities that represent a particular software function that a Network Integrity cartridge performs at run time. A Network Integrity cartridge usually contains multiple actions.

At run time, when an action is deployed to Network Integrity (by deploying a Network Integrity cartridge from Studio, or by using the Oracle Cartridge Deployer), an action is implemented as a J2EE Message Driven Bean (MDB).

Actions are of different types:

- **Discovery Action**: used for discovering data, typically from a network, and persisting the discovered data in the Results Model using POMS entity managers.
- **Import Action**: used for importing data, typically from an inventory system, and persisting the inventory data in the Results Model using POMS entity managers.
- **Assimilation Action**: used for post-processing previously discovered data, and persisting the data in the Results Model using POMS entity managers. The assimilation action cannot produce import results.
- **Discrepancy Detection Action**: used for finding discrepancies between discovered entities and imported entities.
- **Discrepancy Resolution Action**: used for fixing discrepancies in an external system, or a network.

To create an action, see the Network Integrity Studio online Help.

**Actions and Processors**

An action performs a certain function that is supported by a Network Integrity cartridge. To implement this function, a processor is introduced to implement an atomic sub-function, which is part of the functions that this action performs. For example, an SNMP discovery action at least has a processor that carries out SNMP polling on network devices and another processor that models the discovered raw SNMP data into Results Model and persists it using POMS entity managers.

An action contains one or more processors. Each processor is responsible for an atomic function. By chaining the processors inside an action, the action can perform a complex function, such as discovering a network, importing an inventory system, assimilating discovered data, or detecting and resolving discrepancies.

*Figure 4–1* shows a sample action, containing three processors.
When an action is invoked, the processors are executed in the sequence they were placed inside this action. The code-generated Action Controller controls execution. In the example shown in Figure 4–1, the three processors are executed in the following sequence:

- Processor A1
- Processor A2
- Processor A3

See "Working with Processors" for more information about processors.

About the Processors Tab

The Processors tab in the Action Editor enables you to:

- Add a processor to an action. For more information, see the Network Integrity Studio online Help.
- Create and configure For Each components within an action. For more information about For Each components, see "About For Each Processors".
- Move the processor up or down to change the execution sequence of the processors. For more information, see the Network Integrity Studio online Help.
- Remove a processor from the action. For more information, see the Network Integrity Studio online Help.
- Add conditions to apply to the action. For more information, see "Setting Conditions in Network Integrity".
Adding an Existing Action

An action can contain another action. This is done by adding an existing action to a new action. This extensibility is done using Network Integrity Design Studio.

When an action contains another action, it replicates the other action's processors and executes the processors in the same sequence. The execution sequence of the processors added from another action cannot be changed.

New processors can be inserted in between if the overall sequence of the imported processors from another action remains the same.

To add an existing action:

1. Select the Processors tab in the Action Editor.
2. Click Add.
   A dialog is displayed asking you to select an action or processor to add.
3. From the list of available actions, select an action to add.
4. Click OK to add the action.

Figure 4–2 shows the composition of an action.
Action B contains two processors (processor B1 and processor B2), and an action (action A is the sample action shown in Figure 4–1).

In this example, action B actually contains five processors. The sequence of the processors from action A cannot be changed in action B. However, new processors can be inserted between the processors from action A.

For example, the Cisco SNMP cartridge contains a Discovery action, which extends the Discovery action from the MIB-II SNMP cartridge.

Figure 4–3 shows the processors contained inside the Cisco SNMP Cartridge.
This discovery action contains Discover MIB II SNMP as the imported action. By importing the MIB II SNMP Discovery action, Cisco SNMP Discovery action automatically gets the MIB II discovery functions (logical device discovery) provided by the productized MIB-II SNMP cartridge.

In addition, the Cisco SNMP discovery action discovers physical devices (through Cisco SNMP Physical Collector processor and Cisco SNMP Physical Modeler processor), modeling the logical side (through the Cisco SNMP Logical Collector Processor and Cisco SNMP Logical Modeler processor).

**Generated Action MDB and Controller**

Every action becomes a J2EE Message Driven Bean (MDB) at run time. The Controller controls the execution sequence of the processors inside an action.

Both the Action MDB and Controller classes are code-generated. No further Java coding is necessary for either the MDB or the Controller class. These two classes are transparent to a Network Integrity cartridge developer using Design Studio. At design time, the cartridge developer should not have to implement any Java code for an action because all required Java implementations for actions are code-generated.

The generated Action MDB and Controller classes can be found at the following directory:

```
Studio_Workspace\NI_Project_Root\generated\src\Project_Default_Package\Action_Type\Action_Implementation_Prefix
```

where the elements on the path are defined as follows:

- **Studio_Workspace**: Eclipse Workspace root
- **NI_Project_Root**: Network Integrity project root
- **Project_Default_Package**: The default package configured at the cartridge editor
- **Action_Type**: Select from the available action types:
  - assimilationactions
  - detectionactions
  - discoveryactions
  - importactions
- resolutionactions

  Action_Implementation_Prefix: action implementation prefix in lowercase.

The generated MDB class is named: ActionNameMessageDrivenBean.java.

The generated Controller class is named ActionNameMessageDrivenBeanController.java.

During design time, compilation errors or warnings against this Java class might occur. These errors and warnings are cleared after properly implementing and configuring the action (and its processors).

Figure 4–4 shows the directory that contains the generated MDB and Controller classes.

**Figure 4–4  Generated MDB and Controller Class Directory**
About Conditions in Actions

Actions can contain conditions. By creating conditions and applying conditions to processors, you can dynamically control which processors are executed inside an action.

See "Setting Conditions in Network Integrity" for more information about conditions.

About Model Collections in Actions

Actions can contain model collections. A model collection is a collection of specifications.

See "Model Extension Using Specifications" for more information.

Adding a model collection to an action enables the generation of the Specification Helper classes (when Studio is building the action). These classes are by the action for modeling the discovered data into the Oracle Communications Information Model and persisting it using POMS entity managers.

If an action is imported into another action in a different cartridge, the Network Integrity packager uses the model collections to determine how to build the specification DAO files so that all the model collections (from both the imported action and the current action) are included.

See "Model Collections" for more information about model collections.

About For Each Processors

An action can contain a For Each processor, which is added when configuring processors using the Processors tab. The Action Controller sets the execution sequence of the processors based on the order in which the processors are configured. Usually a processor is invoked only once, and when it has executed, the Controller invokes the next processor, until all processors in an action are invoked.

However, one or more processors may be executed repeatedly. For example, when importing an inventory system, it is typical to first get a list of devices from the inventory system, then go through the list of devices and import each device singly into Network Integrity. In this example, the processor importing a single device is repeatedly executed for all the devices in the returned device list. You can use For Each processors to create a loop, containing one or more processors, to repeatedly execute the processors. Design Studio for Network Integrity supports nested For Each processors.

Creating For Each Processors

A For Each processor expects a collection as the input parameter so that it can iterate through the collection and, for each object in the collection, invoke the processors inside the loop. There must be a preceding processor that outputs an array or a Java object that implements java.lang.Iterable (for example, java.util.List) as an output parameters to create a For Each processor.

Figure 4–5 displays a processor, GetAllDevices, which outputs a list, deviceList in the action, DemoImportAction.
The next step is to create a For Each processor, called *For Each device in deviceList*. To create a For Each processor, see the Network Integrity Studio online Help.

*Figure 4–6* displays the new *For Each* processor.
Next, create a processor, *ImportDevice*, which runs inside the *For Each* processor to import a single device into Network Integrity. To create a processor, see the Network Integrity Studio online Help.

After creating the processor, click **Move Up** to move this processor inside the *For Each* processor.

*Figure 4–7* shows the resultant processor implementation.
Expand the *For Each* processor to see *ImportDevice* inside the *For Each* loop. For each device from the input device list, *ImportDevice* imports it into Network Integrity.
In Oracle Communications Network Integrity, processor entities are the building-blocks for actions, as they implement atomic sub-functions for actions.

For example, an SNMP processor is included in an action to poll network devices; a modeler processor is included in an action to model raw SNMP data from a network device and add it to a database. Combined, these two processors comprise a discovery action that polls SNMP-enabled network devices and persists the modeled SNMP data.

By adding multiple processors to an action, the action performs several complex function by executing the processors according to the sequence in which they were added to the action.

Processors are of different types:

- **Discovery Processor**: part of a discovery action.
- **Import Processor**: part of an import action.
- **Assimilation Processor**: part of an assimilation action.
- **Discrepancy Detection Processor**: part of a discrepancy detection processor action.
- **Discrepancy Resolution Processor**: part of a discrepancy resolution action.
- **File Transfer Processor**: used to retrieve files from local or remote directories. For more information, see *Network Integrity File Transfer and Parsing Guide*.

- **File Parsing Processor**: used to parse data retrieved by the File Transfer processor so that the data is available to other processors. For more information, see *Network Integrity File Transfer and Parsing Guide*.

Unlike actions, processors are not visible in Network Integrity.

To create a processor, see the following:

- **Creating Processors**

To configure a processor, see the following:

- **Configuring Processors**

To configure the input and output parameters for the processor, see the following:

- **About Context Parameters**

To configure property groups and properties for a processor, see the following:

- **About Properties and Property Groups**

To view an outline of code generation for processors, see the following:
Creating Processors

You can create a processor independently or create a processor in the process of adding it to an action. The latter method is recommended because it automatically adds the processor to the list of processors that the action uses. And it also ensures that you can create only the supported types of processors for the current action.

To create a processor, see the Design Studio Help.

Configuring Processors

The main steps in configuring a processor using the processor editor in Design Studio include:

- Using the Properties tab to define properties that are passed to the processor.
- Using the Context Parameters tab to define the processor’s inputs and outputs.
- Using the Details tab to specify the implementation class.

After you configure an action and its processors, complete the action by coding the implementations for the processors.

About Context Parameters

The processor editor has a Context Parameters tab that you can use to configure the input and output parameters for the processor. Both input and output parameters are optional for a processor.

For extensibility, configure the processor to produce an output parameter that is available to other processors to continue data processing. Typically, the output parameter should be the Oracle Communications Information Model entity that the processor models: for example, LogicalDevice or PhysicalDevice.

See Oracle Communications Information Model Reference and Network Integrity Information Model Reference for further information about the Information Model.

After adding input and output parameters for the processor using Oracle Communications Design Studio, these parameters appear in tabular format in the Context Parameters area of the processor editor. Design Studio generates the request and response Java classes based on the input and output parameters.

About Properties and Property Groups

A property group is a logical container configured on a processor. A property group can be added to multiple processors. Property group names must be unique within a processor.

Properties are added to property groups and are assigned property values to pass to the processor, either hard-coded or at run time.

Property groups do not inherently pass any values to the processor other than the values belonging to its properties.
Property groups and properties are configured on processors on the **Properties** tab of the processor editor.

Property groups can be configured as Managed groups, where the values for the properties it contains can be set at run time using the MBean interface. See *Network Integrity System Administrator’s Guide* for more information.

Property groups can be configured as Map groups, where the property group produces a simplified API for properties that are used as maps.

A Java class is generated for the property group so that you can extend a cartridge to access the property values it contains using a generated interface.

A property consists of a name-value pair that is passed to the processor through the property group. Depending on how the property group is configured, the property value is either hard-coded, or provided at run time through the MBean interface.

Property names must be unique within the property group.

Properties can be configured with the following options:

- Property values can be set using a cartridge model variable, where you can specify the value of the variable at deployment time. To set a property value with a cartridge model variable, the value string must begin and end with a percentage (%) symbol, as in the following example:

```
%Property_Value%
```

- Properties can be configured as Secret values, to pass encrypted values at deployment time using the MBean interface. The property value must be encrypted before it can be entered in the MBean interface. See *Network Integrity System Administrator’s Guide* for more information.

For more information on adding property groups to a processor, adding properties to a property group, and setting cartridge model variables, see the Design Studio Help.

---

**About Generated Code**

This section describes code generation for processors in Network Integrity:

- **About the Location for Generated Code**
- **About the Processor Interface**
- **About the PropertyGroup and Properties Classes**

**About the Location for Generated Code**

Design Studio code-generates the relevant Java classes for the processor. The generated code is located at:

```
Studio_Workspace\NI_Project_Root\generated\src\Project_Default_Package\Processor_Type\Processor_Implementation_Prefix
```

where:

- **Studio_Workspace** is the Eclipse Workspace root
- **NI_Project_Root** is the Network Integrity project root
- **Project_Default_Package** is the default package configured at the cartridge editor
- **Processor_Type** is run time following action types:
  - discoveryprocessors
About Generated Code

- importprocessors
- assimilationprocessors
- detectionprocessors
- resolutionprocessors
  - Processor_Implentation_Prefix is the action implementation prefix in lowercase.

About the Processor Interface

Every processor has a generated interface. The generated processor interface class is named \texttt{Processor\_NameProcessorInterface.java}.

In general, the generated processor interface has the \texttt{invoke} method defined. The interface has two forms of \texttt{invoke} methods, depending on whether there is an output parameter defined for the processor.

```java
// Signature for processor which does not have output parameters
public void invoke(<Processor\_Specific\_Context> context,
                   ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
}
```

```java
// Signature for processor which has output parameters
public ExampleProcessorResponse invoke(<Processor\_Specific\_Context> context,
                                        ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
    return null;
}
```

The generated processor interface has a slightly different signature, depending on the type of processor: for example, \texttt{Processor\_Specific\_Context} differs between processor types. See individual chapters on specific processors for more information.

About the PropertyGroup and Properties Classes

A properties class is always code-generated for the processor, whether the processor has property groups and properties configured or not. The properties class is used as an input parameter for the constructor of the generated request class.

The generated properties class is named \texttt{Processor\_NameProcessorProperties.java}.

The generated properties class has a public method, \texttt{String[] getValidProperties()}. This method returns a string array that contains a list of valid property group names configured for this processor. If the processor has no property groups configured, this method returns an empty array.

If the processor has property groups and properties configured, for each property group a PropertyGroup class is code-generated.

The generated PropertyGroup class is named \texttt{PropertyGroup\_NamePropertyGroup.java}.

The generated PropertyGroup represents the configured property group and all of its properties. The generated properties class has the getter methods to get each PropertyGroup directly, and has all the setter methods to modify the property values.

The generated PropertyGroup class has a public method, \texttt{String[] getValidProperties()}. This method returns a string array that contains a list of valid properties names configured for this property group. If the property group has no property configured, this method returns an empty array.
If the property group is not configured as a Map group, the generated PropertyGroup class provides getter methods for all the properties configured in this property group.

If the property group is configured as a Map group, the generated PropertyGroup class does not provide getter methods for all the properties configured in this property group. Instead, the API for the property group resembles a Java Map, where the property values are retrieved and set using the property name passed as a value.

About Processor Implementation

Implementation of the processor is done in the processor editor using the Details tab. See the Design Studio Help for specific configuration details.

You can click the Implementation Class link to open the Java editor for this implementation Java class. Design Studio auto-generates the skeleton Java implementation class, which implements the processor interface with an empty implementation method.

You must decide whether to complete implementing the method. Sometimes, if the processor was changed later (for example, by adding output parameters or removing parameters) the implementation class displays a compiling error. This is expected because the skeleton implementation class is regenerated. You must modify the implementation class to match the changed processor interface.

For information about how to implement a processor, see the individual processor section.

About the Processor Finalizer

When a processor deals with resources (for example, sockets and files), it is necessary to clean up the resources used or created while the processor executes. Using a finalizer on the processor ensures that the used or created resources get cleaned up, whether the action fails or is successful. When implemented, the finalizer cleans up the resources used or created by the processor. It is not mandatory to implement the finalizer if the processor does not deal with a resource, or if the resource is used only within the processor (in which case the processor implementation should make sure the local resource is closed properly). The processor must implement the finalizer if the processor allocates a resource that is to be output for use by other processors.

Finalizers that are not inside a For Each loop are called by the action controller class (code-generated) before it completes. Finalizers that are inside a For Each loop are called by the action controller class at the end of the For Each loop. In all cases, finalizers are called in the reverse order to which they are registered (finalizers registered first are called last; finalizers registered last are called first).

ProcessorFinalizer

The processor implementation class must implement the interface oracle.communications.sce.integrity.sdk.processor.ProcessorFinalizer to have the action controller clean up the resources that are used or created by the processor. If a processor does not use or create a resource, it does not implement the ProcessorFinalizer interface.

The processor defines only one method:

```java
public void close(boolean failed);
```
The processor that implements the ProcessorFinalizer interface must implement this method to close all the resources used or created during the execution of this processor. This method takes an input parameter as Boolean. If there is an exception during the execution of the processors, the action controller calls the finalizer by passing True to this method; otherwise the action controller calls the finalizer by passing False to the method, in the successful case. The processor might implement the close logic differently for both successful and failed scenarios: for example, if it is a failed scenario, the close method might log an error message before closing the resources.

The following code shows how to implement the ProcessorFinalizer for a sample processor:

```java
public class SampleProcessorImpl implements SampleProcessorInterface, ProcessorFinalizer {
    public SampleProcessorResponse invoke(SampleProcessorRequest request)
        throws ProcessorException {
        // Implement the Processor here...
    }

    public void close(boolean failed) {
        if(failed) {
            // something is failed, log extra error message here.
        }
        // close the InputStream here.
        try {
            myInputStream.close()
        } catch(IOException ioe) {
            // log the IOException here...
        }
    }
}
```

**About Memory Considerations**

The action controller class calls the finalizers for both successful and failed scenarios. The finalizers that are not inside a For Each loop do not begin until the end of the action. The finalizers that are inside a For Each loop do not begin until the end of the loop. When a processor that implements the ProcessorFinalizer completes the execution, it is still in the scope of the action. The processor does not get purged by the garbage collector to release the memory.

If a processor implements the ProcessorFinalizer, it is a good practice to limit the number of member variables for that processor and ensure that the processor is not using a large amount of memory. If the processor uses a lot of memory, it is a good practice to release the memory as soon as it is no longer required. For example, if a processor is using a large HashMap, and it also implements the ProcessorFinalizer, the processor should clear the contents of the HashMap when it is done using it and assign the null pointer to this HashMap.
The discovery action is used to discover data, typically from the network, and to persist the discovered data in the Results Model. The discovery action accesses the network using a variety of technologies and protocols; for example: SNMP.

Because SNMP is such an important protocol for network discovery, Network Integrity provides specific features to allow streamlined development of SNMP network discovery cartridges within Design Studio for Network Integrity.

See “Working with SNMP Processors” for further information.

About Supported Processors

The discovery action supports the following processor types:

- **Discovery processor.** The discovery processor is a general processor, which has no concrete implementation; it can be implemented using Design Studio to discover network data through various technologies and protocols (for example, TL1 and CORBA) or to model discovered data and persist the data in the Results Model. See "Configuring a Discovery Processor".

- **SNMP processor.** The SNMP Processor is a completely implemented processor that polls SNMP-enabled network devices using SNMP. See "Configuring the SNMP Processor".

- **File Transfer processor.** For information about the File Transfer processor, see *Network Integrity File Transfer and Parsing Guide*.

- **File Parsing processor.** For information about the File Parsing processor, see *Network Integrity File Transfer and Parsing Guide*.

Creating a Discovery Action

To create a discovery action:

1. In Design Studio, from the **Studio** menu, click **Show Design Perspective**.
2. Select **New** from the **Studio** menu, then select **Integrity**, then select **Discovery Action**.
   The Discovery Action Wizard appears.
3. Refer to Network Integrity Studio online Help for further information on creating an action.
Configuring a Discovery Action

This section covers those fields that are specific to the discovery action. For information about configuring common fields for all action types, see "Working with Actions" or Design Studio Help.

About Address Handlers

You use the Details tab in the Design Studio action editor to configure a discovery action with an optional Address Handler. To add an existing address handler to a discovery action, see Network Integrity Studio online Help.

If the discovery action is configured with an Address Handler, the Address Handler validates the addresses configured in the scope for the discovery action; depending on the implementation of the Address Handler, the Address Handler may also expand a range of addresses when saving a scan. If the address specified in the scope is not valid, the scan is not saved.

The Address Handler is an optional component for the discovery action. A discovery action without an Address Handler does not have the addresses validated (or expanded if it is a range) when creating a scan. In this case, every address configured in the scope is treated as a single valid address. This approach is not recommended, because an invalid address fails a network discovery at run time. A properly implemented Address Handler catches this kind of error at scan configuration time.

About Result Categories

A discovery action must be configured with a valid result category, which is a mandatory field for a discovery action. If no result category is configured, a validation error is raised and an error marker displayed against the discovery action.

The following is a sample validation error message for a discovery action without result category configured:

*Missing - Action has not specified a result category. At least one result category must be specified.*

To add a result category to a discovery action, see Network Integrity Studio online Help.

Model validation is required to ensure there is no overlap in the registration of results from different sources.

To add result categories:

1. Click the Details tab of the action in a configuration editor.
2. In the Results Categories area, click Add.
3. The Create Results Categories dialog box appears.
4. In the Name field, enter a name for the result category. Specify a proper name for the result category to be created. In Figure 6–1, the name Device is provided.
5. In the Description field, you can add information about the result category.
6. Click OK.
7. From the File menu, select Save.

Figure 6–1 displays a result category, Device.
After successfully building a discovery action in Design Studio (see "Building and Packaging Cartridges"), deploy the cartridge to Network Integrity (see "Deploying and Undeploying Cartridges").

When the cartridge containing the discovery action is successfully deployed to Network Integrity, log on to the Network Integrity user interface and configure a scan using the deployed discovery action.

The recently deployed discovery action is available in the Scan Action list when creating a scan configuration. See Network Integrity online Help for further information about creating a scan.

Figure 6–2 displays a discovery action called Discover Sample Device.
About UI Parameters

You can configure UI parameters for a discovery action. Usually, a discovery action requires configuration to establish a connection to network devices.

For example, the following parameters could be considered:

- **Port**: the port number that a discovery command is sent to
- **Username**: the user name to make the connection
- **Password**: the password to make the connection

When a scan is created using Discover Sample Device (see "About the Discovery Action in the Network Integrity User Interface"), the **Scan Action Parameters** section on the Create Scan page is filled with SNMP UI parameters.

**Figure 6–3** displays the **Scan Action Parameters** section for Discover Sample Device with SNMP UI parameters configured.
To make configuration items available when creating the scan, appropriate UI parameters must be configured.

Use the UI Parameters tab in the Design Studio action editor to configure a discovery action with appropriate UI parameters. Add new UI parameters or select existing ones. See Network Integrity online Help for further information.

See also "Working with UI Parameters" for further information.

### Creating Discovery Processors

There are several ways to create a discovery processor. However, the best way is to create it using the Processor tab for the discovery action. This method automatically adds the processor to the list of processors that the action uses. And it also ensures that only the supported type of processor is created for the current action.

To create a processor, see Network Integrity Studio online Help.

---

**Note:** Discovery actions support two types of processor: discovery processor and SNMP processor. In the Processor Wizard, the **Type** field is enabled. Ensure **Discovery Processor** is selected when creating a discovery processor.

---

### Configuring a Discovery Processor

See "Working with Processors" for more information on configuring a discovery processor in Design Studio.

### Implementing a Discovery Processor

Configuration of the discovery action and its discovery processors results in the generation of many deployment artifacts. However, you must supply implementations for the discovery processors.

The implementation needs to implement the `invoke` method. Two forms of this method are shown:

```java
// Signature for processor which does not have output parameters
public void invoke(DiscoveryProcessorContext context, ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
}

// Signature for processor which has output parameters
public ExampleProcessorResponse invoke(DiscoveryProcessorContext context, ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
    return null;
}
```

The parameters and return type of the `invoke` method are:

- **Processor_Name**ProcessorResponse: this is the return type, for processors that have output parameters. For processors that do not have output parameters, the return type is `void`. This class is generated by Design Studio. It is a value object containing values for each of the processor's output parameters. For processors that have output parameters, the `invoke` method must create a ProcessorResponse object, set it values and return the ProcessorResponse object.
Processor_NameProcessorRequest: this is a value object that has the following getters:

- If UI Parameters have been specified for the discovery action, there is a getter that returns a UI Parameters value object.
- If properties have been defined for the discovery processor, there is a getter that returns a Processor_NameProcessorProperties value object.
- There is a getter for each input parameter that is defined for the processor.
- There is a getter method called `getScopeAddress`. This method returns the scope address configured for this discovery action.

This class is generated by Design Studio.

DiscoveryProcessorContext context: this is an SDK type, which has the following methods:

- `getActionName`: returns the name of the action that the processor is executing under.
- `getProcessorName`: returns the name of the processor.
- `persistResults`: causes POMS objects to be flushed to the database. This helps to reduce memory consumption. See "About Persist Results" for more information.
- `addToResult`: this is used to add a graph of POMS objects to the database under a result group. This method takes three parameters:
  * String `resultGroupName`: this is the name of a result group under which the results are persisted.
  * String `resultGroupType`: this is the type of the result group under which the results are persisted. This should match a category defined on the action.
  * DiscrepancyEnabled result: this is the root of result object graph to be persisted.
- `getResultGroup`: used to get an existing result group from your current scan if you must access the graph of POMS objects previously added to a result group. This method takes two parameters:
  * String `resultGroupName`: this is the name of a result group under which the results are persisted.
  * String `resultGroupType`: this is the type of result group under which the results are persisted. This should match a category defined on the action.

**Implementation Code Sample**

The following Java code snippet demonstrates how to implement `invoke` for a discovery processor, and how to add results to the result group using the method, `addToResult`.

```java
public SampleProcessorResponse invoke(
    DiscoveryProcessorContext context,
    SampleProcessorRequest request) throws ProcessorException {
    SampleProcessorResponse modelerResponse = new SampleProcessorResponse();
    SampleDevice device;

    // Get the input Sample Response Document from the Request.
```
// This input response document is used to model the sample device.
SampleResponseType response = request.getSampleResponseDocument();

try {
   // Make the Sample Device
   device = makeSampleDevice(response);
   // Add the device to the result group "Device", which matches
   // the result category configured in the Discovery Action.
   context.addToResult(device.getName(), "Device", device);
   modelerResponse.setSampleDevice(device);
}

} catch (Exception e) {
   // Handle exception here...
}

return modelerResponse;
This chapter describes the SNMP (Simple Management Network Protocol) processor, which is one of two processor types supported by the discovery action in Network Integrity.

The SNMP Processor is a completely implemented processor that polls SNMP-enabled network devices using the SNMP protocol.

The other supported processor is the discovery processor, which is a general processor, and has no concrete implementation; it can be implemented using Design Studio to discover network data through various technologies and protocols (for example, TL1 and CORBA) or to model discovered data and persist the data in the Results Model. See “Configuring a Discovery Processor”.

See “Working with Discovery Actions and Processors” for general information about discovery actions and supported processors.

Creating an SNMP Processor

There are several ways to create an SNMP processor. However, the best way is to create it using the Processor tab for the discovery action. This method automatically adds the processor to the list of processors that the action uses. And it also ensures that only the supported type of processor is created for the current action.

To create a processor, see Network Integrity Studio online Help.

---

**Note:** Discovery actions support two types of processor: discovery processor and SNMP processor. In the Processor Wizard, the Type field is enabled. Ensure SNMP Processor is selected when creating an SNMP processor.

---

When you create an SNMP processor and add it to a Discovery action, a code-generated SNMP UI parameter is automatically added to this Discovery action (available in the “UI Parameters” tab in the Discovery Action Editor).

**Note:** Do not modify this code-generated SNMP UI parameter in the Data Dictionary Editor. If you must extend the SNMP UI parameter, create a UI parameter instead of modifying the code-generated SNMP UI parameter.
Configuring the SNMP Processor

The SNMP Processor is a code-generated Discovery processor with the complete implementation. The SNMP Processor polls an SNMP-enabled network device by sending a request containing a list of OIDs (and other relevant SNMP configurations) to the SNMP Adapter, and receiving the response containing a strongly-typed XML document, which represents the raw SNMP results.

There is no coding required for the SNMP processor. The Processor Interface, Request/Response, Properties, and the relevant helper classes of an SNMP Processor are all code-generated and fully implemented.

The only configuration required for the SNMP Processor is to configure the list of polled OIDs. Before configuring the OIDs for the SNMP Processor, the MIB directory must be properly specified for the Network Integrity preference. If the MIB directory is not properly specified in the preference, you cannot configure the SNMP processor.

See "Setting Preferences in Network Integrity" for more information.

When the list of polled OIDs has been configured, the selected MIBs are available.

Figure 7–1, displays two MIBs, ATM-MIB and RFC1213-MIB, as selected.

![Figure 7–1 Configuring the SNMP Processor](image)

About the Generated Implementation and XML Beans

The SNMP processor is a completely code-generated discovery processor. Along with the usual discovery processor implementations (see "Implementing a Discovery Processor"), Design Studio also generates the strongly-typed SNMP XML response document schema based on the OIDs configured for the SNMP processor.
Creating an SNMP Processor

The generated SNMP XML response document schemas are available at the following directory:

`Project_Root\generated\SNMP_Processor_Name_snmpdiscoveryprocessor`.

Under this directory, the following sub-directories exist:

- **lib**: this directory contains the compiled XML Beans JAR file for the strongly-typed SNMP XML response document schemas
- **snmpClasses**: this directory contains the XML Beans Java classes for the strongly-typed SNMP XML response document schemas
- **snmpSchemas**: this directory contains the generated strongly-typed SNMP XML response document schemas
- **snmpSrc**: this directory contains the compiled XML Beans Java source for the generated strongly-typed SNMP XML response document schemas.

It is recommended to first look at the schemas generated in this directory to understand how to access the compiled XML Beans object for the SNMP response document.

The remaining implementations for the SNMP Processor are at the following directory:

`Studio_Workspace\NI_Project_Root\generated\src\Project_Default_Package\snmpdiscoveryprocessors\SNMP_Processor_Implmentation_Prefix`

The SNMP Processor always has an output parameter, which is the SNMP XML response document (XML Beans object). This is available in the Response class for the SNMP processor.

**Supporting New MIBs**

When the productized Network Integrity cartridges are imported into Design Studio (see "Exporting and Importing Cartridges"), Network Integrity cartridges are bundled with a set of MIB files, which is the same set of MIB files bundled with the SNMP Resource Adapter (see "JCA Resource Adapters").

If you must create a Network Integrity cartridge to poll certain MIB OIDs for certain specific devices, which are not part of the bundled MIB files, you must get the MIB file (or set of MIB files) that has the definitions of those MIB OIDs required to implement the new cartridge.

The new MIB files must be manually copied to the MIB directory configured in the Design Studio preference (see "Setting Preferences in Network Integrity"). After the new MIB files are copied to the MIB directory, the new MIB files are available to be loaded in Studio. There is no need to restart Design Studio.

---

**Note:** The MIB files on Studio and on the SNMP resource adapter must match. See "JCA Resource Adapters" for information about supporting new MIBs for the SNMP resource adapter.
Import actions are used to import data from an inventory system into Network Integrity. The data is stored in the Oracle Communications Information Model representation and is flagged as having come from the inventory system. The Network Integrity GUI displays and reports on the data discovered by an import action. The data can also subsequently be processed by Discrepancy actions that compare network-discovered data to inventory-discovered data, and reports differences between them.

Import actions are edited in Design Studio. As a result of the editing, Studio generates most of the required deployment artifacts. However, you must supply some Java implementation. After this is done, and all error problems are cleared, and if the import action is not abstract, Studio automatically packages the action into a cartridge Integrity ARTifact (IAR) file that can be easily deployed into the Network Integrity server. Then, on the Network Integrity server, an import scan can be created and executed, and the scan results viewed or reported on.

To create an import action, see the following:
- Creating Import Actions

To configure an import action, see the following:
- Configuring Import Actions

To create an import processor, see the following:
- Creating Import Processors

To configure an import processor, see the following:
- Configuring Import Processors

To implement an import processor, see the following:
- Implementing an Import Processor

**Creating Import Actions**

You can create import actions using Design Studio. Import actions are created inside a Network Integrity project.

To create an import action:

1. Select **New** from the **Studio** menu, then select **Integrity**, then **Import Action**.

   The Import Action Wizard appears.
2. In the **Project** field, set the value to the previously selected cartridge. You can change the cartridge under which the import action is created by selecting a different cartridge.

3. In the **Name** field, enter a name for the new action.

4. In the **Folder** field, select a location for the action. If you leave the **Folder** field blank, the action is created in the default location. The folders can be found inside the Import Actions folder of the project tree in the Cartridge view.

5. Click **Finish**.

   Studio creates the new action and displays it in the action editor. The Details page is displayed. See "Details Page" for more information on configuring this page.

---

**Configuring Import Actions**

The import action editor contains several tabbed pages. The following links take you to reference topics for the pages used to define an action:

- Details Page
- Processors Page
- UI Parameters Page
- Model Page
- Conditions Page

---

**Details Page**

The Details page contains general settings for the action, as shown in **Table 8–1**.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lets you provide a short description of the action.</td>
</tr>
<tr>
<td>Implementation Prefix</td>
<td>Takes the value of the component name by default. This field cannot be changed. This field is used by Design Studio when creating some deployable artifacts.</td>
</tr>
<tr>
<td>Abstract</td>
<td>Lets you specify whether the action is abstract. If the action is abstract, it can only be used as a component within another action. If the action is not abstract, it can be deployed and a scan can be executed from it.</td>
</tr>
<tr>
<td>Result Categories</td>
<td>A multi-valued field that lets you define the result categories for the import action. A result category classifies the results of a scan. Different parts of a scan can be placed into different result categories.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Lets you add any other information about the action.</td>
</tr>
</tbody>
</table>

---

**Processors Page**

The Processors page is used to define the processors included with the action.

See "About the Processors Tab" for more information about the tasks carried out using the Processors tab.
The only processor type that can be created and used for an import action is the import processor.

**UI Parameters Page**

Use this page to define the UI Parameters for the import action. UI Parameters are displayed as editable fields in the Network Integrity GUI. The UI Parameters are passed to the implementation code for the import action processors.

See "Using UI Parameters in Actions" for more information about UI parameter configuration.

**Model Page**

Use the Model page to specify the model collections that the import action uses. The model collection defines the parts of the Information Model and the dynamic extensions that are related to the model collection. In effect, the Model tab specifies which parts of the Information Model are used by the import action.

See "About Model Collections in Actions" for more information about model collections.

**Conditions Page**

The Conditions page allows conditions to be defined. Conditions can be associated with processors to cause conditional execution of the processor.

See "About Conditions" for more information about conditions.

**Editing Import Actions**

This section provides an outline of the main steps involved in editing an import action.

1. From the import action editor, select the **Details** page.
   a. Define the result categories to be produced by the import action.
   b. Choose whether the import action is abstract
   c. Enter a short description in the **Description** field
   d. Enter a more complete description in the **Documentation** field

2. On the **UI Parameters** page, define the parameters that the user enters in the Network Integrity GUI when running a scan.

3. On the **Model** page, define the Model Collections that the import action uses. The Model Collections define a subset of the Information Model and specifications.

4. On the **Processors** page, define the flow of processors that comprise the import action.
   a. If conditional logic is required, use the **Conditions** tab to define conditions for the processors. Use the **Processor** tab to associate the conditions with the processors.
   b. Define iterating logic using For Each constructs.

      See "About For Each Processors" for more information on the For Each construct.
Creating Import Processors

There are several ways to create an import processor. However, the best way is to create it using the Processor tab for the import action. This method automatically adds the processor to the list of processors that the action uses. And it also ensures that only the supported type of processor is created for the current action.

To create an import processor:

1. From the Processor tab of the import action, in the action processors section, click Add.
   The Import Processor Wizard appears.

2. Do not change the value in the Project field, as this field should already be set to the correct cartridge.

3. In the Name field, enter a name for the new processor.

4. In the Folder field, select a location for the processor. If you leave the Folder field blank, the action is created in the default location. The folders can be found inside the Import Processors folder of the project tree in the Cartridge view.

5. Click Finish.
   Studio creates the new processor and displays it in the action editor. The Details page is displayed.

6. Click Save to save the import action.

7. Double-click the new import processor in the Actions Processors list to open the new import processor editor.

Configuring Import Processors

The import processor editor contains several tabbed pages.

The following link takes you to reference topics for the pages used to define an import processor:

- Details Page
- Context Parameters Page
- Properties Page

Details Page

The Details page contains general setting for the processor, as shown in Table 8–2.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lets you provide a short description of the processor.</td>
</tr>
<tr>
<td>Implementation Class</td>
<td>Lets you specify the implementation class for the processor. Design Studio automatically generates an interface class, which the implementation class must implement. You can select an existing class, or create a class, by selecting the Implementation Class link. If you use the link, the implementation class is created to match the processor interface.</td>
</tr>
</tbody>
</table>

Table 8–2 Details Page
Implementing an Import Processor

Working with Import Actions and Processors

Context Parameters Page

The Context Parameters page is used to define the processor's inputs and outputs.

See "About Context Parameters" for more information about the tasks done using the Context Parameters page.

Properties Page

Use this page to define the properties for the import processor. Properties are name value pairs that are passed to the processor.

See "About Properties and Property Groups" for more information about properties configuration.

Editing Import Processors

This section provides an outline of the main steps involved in editing an import processor.

1. Define properties to be passed to the import processor using the Properties page.
2. On the Context Parameters page, define the inputs and outputs for the import processor.
3. Use the Details page to specify the implementation class.

When the editing of the import action and processors is completed, complete the import action by coding the implementations for the import processors.

Implementing an Import Processor

Many deployment artifacts for the import action and its processors are generated automatically while editing. However, you must supply implementations for the import processors using the invoke method.

Two forms of this method are shown in the following code fragments:

```
// Signature for processor which does not have output parameters
public void invoke(DiscoveryProcessorContext context,
        ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
}

// Signature for processor which has output parameters
public ExampleProcessorResponse invoke(DiscoveryProcessorContext context,
        ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
    return null;
}
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
</table>
| Implementation Prefix | You cannot change this field. It is set automatically, and is based on the name of the import processor.  
|                   | This field’s value is used by Design Studio when creating some deployable artifacts.                                                      |
| Documentation    | Lets you add any other information about the processor.                                                                                     |
The parameters and return type of the `invoke` method are:

- **Processor_Name**ProcessorResponse: this is the return type, for processors that have output parameters. For processors that do not have output parameters, the return type is void. This class is generated by Design Studio. It is a value object containing values for each of the processor’s output parameters. For processors that have output parameters, the `invoke` method must create a ProcessorResponse object, set its values and return the ProcessorResponse object.

- **Processor_Name**ProcessorRequest: this is a value object that has the following getters:
  - If UI Parameters are specified for the import action, there is a getter that returns a UI Parameters value object
  - If properties are defined for the import processor, there is a getter that returns a `Processor_Name`ProcessorProperties value object
  - There is a getter for each input parameter that is defined for the processor
  - There is a getter method called `getScopeAddress`. This method is not useful for import processor implementation. Instead, the inventory system address and authentication information should be retrieved using the POMS API. See "POMS SDK" for more information.

This class is generated by Design Studio.

- **DiscoveryProcessorContext context**: this is an SDK type that has the following methods:
  - `getActionName`: returns the name of the action that the processor is executing under
  - `getProcessorName`: returns the name of the processor
  - `persistResults`: causes POMS objects to be flushed to the database. This helps to reduce memory consumption. See "About Persist Results" for more information.
  - `addToResult`: this is used to add a graph of POMS objects to the database under a result group. This method takes three parameters:
    * String resultGroupName: this is the name of a result group under which the results are persisted
    * String resultGroupType: this is the type of the result group under which the results are persisted. This should match a category defined on the action.
    * DiscrepancyEnabled result: this is the root of result object graph to be persisted.
  - `getResultGroup`: used to get an existing result group from your current scan if you must access the graph of POMS objects previously added to a result group. This method takes two parameters:
    * String resultGroupName: this is the name of a result group under which the results are persisted.
    * String resultGroupType: this is the type of result group under which the results are persisted. This should match a category defined on the action.
Assimilation actions perform additional processing on existing Network Integrity network data to derive additional, often higher level, information from the data. For example, an assimilation action might be used to derive connectivity relationships between endpoints discovered by previous scans.

Assimilation actions are different from other types of scans in that they do not retrieve their data from external sources. Instead, assimilation action scans work on the scan results of other discovery, import, or assimilation scans. When you run an assimilation action scan, the scan selects other scans as inputs to the assimilation scan in the Scope page of the Network Integrity GUI. You can select discovery, import, or other assimilation scans as input.

As with other scan types, the data from assimilation actions is stored in the Oracle Communications Information Model representation. The data from assimilation scans is flagged as having come from the network. The Network Integrity GUI displays and reports on the data discovered by an assimilation action. The data can also subsequently be processed by discrepancy actions, which compare network discovered data to inventory discovered data and report where differences are found.

Assimilation actions are edited in Design Studio. As a result of the editing, Studio generates most of the required deployment artifacts. However, you must supply some Java implementation. After this is done, and all error problems are cleared, and if the assimilation action is not abstract, Studio automatically packages the action into a cartridge Integrity ARtifact (IAR) file, which can be easily deployed into the Network Integrity server. Then, on the Network Integrity server, an assimilation scan can be created and executed, and the scan results viewed or reported on.

To create an assimilation action, see the following:

- Creating Assimilation Actions

To configure an assimilation action, see the following:

- Configuring Assimilation Actions

To create an assimilation processor, see the following:

- Creating Assimilation Processors

To configure an assimilation processor, see the following:

- Configuring Assimilation Processors

To implement an assimilation processor, see the following:

- Implementing Assimilation Processors
Creating Assimilation Actions

You can create assimilation actions using Design Studio. Assimilation actions are created inside a Network Integrity project.

To create an assimilation action:

1. Select New from the Studio menu, then select Integrity, then Assimilation Action.

   The Assimilation Action Wizard appears.

2. In the Project field, set the value to the previously selected cartridge. You can change the cartridge under which the assimilation action is created by selecting a different cartridge.

3. In the Name field, enter a name for the new action.

4. In the Folder field, select a location for the action. If you leave the Folder field blank, the action is created in the default location. The folders can be found inside the Assimilation Actions folder of the project tree in the Cartridge view.

5. Click Finish.

   Studio creates the new action and displays it in the action editor. The Details page is displayed. See "Details Page" for more information on configuring this page.

Configuring Assimilation Actions

The assimilation action editor contains several tabbed pages.

The following link takes you to reference topics for the pages used to define an action:

- Details Page
- Processors Page
- UI Parameters Page
- Model Page
- Conditions Page

Details Page

The Details page contains general settings for the action, as shown in Table 9–1.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lets you provide a short description of the action.</td>
</tr>
<tr>
<td>Implementation Prefix</td>
<td>This field is a read-only field. The value for this field is set automatically by Design Studio and is derived from Assimilation Action's name. This field is used by Design Studio when creating some deployable artifacts.</td>
</tr>
<tr>
<td>Abstract</td>
<td>Lets you specify whether the action is abstract. If the action is abstract, it can only be used as a component within another action. If the action is not abstract, it can be deployed and a scan can be executed from it.</td>
</tr>
</tbody>
</table>
The Processors page is used to define the processors that comprise the action. See "About the Processors Tab" for more information about the tasks done using the Processors tab. The only processor type that can be created and used for an assimilation action is the assimilation processor. The Processors tab also enables you to associate conditions with a processor.

The UI Parameters page

Use this page to define the UI Parameters for the assimilation action. UI Parameters are displayed as editable fields in the Network Integrity GUI. The UI Parameters are passed to the implementation code for the assimilation action processors. See "Using UI Parameters in Actions" for more information about UI parameter configuration.

The Model page

Use the Model page to specify the model collections that the assimilation action uses. The model collection defines the parts of the Information Model and the dynamic extensions that are related to the model collection. In effect, the Model tab specifies which parts of the Information Model are used by the assimilation action. See "About Model Collections in Actions" for more information about model collections.

The Conditions page

The Conditions page allows conditions to be defined. Conditions can be associated with processors to cause conditional execution of the processor. See "About Conditions" for more information about conditions.

Editing Assimilation Actions

This section provides an outline of the main steps involved in editing an assimilation action.

1. From the assimilation action editor, select the Details page.
   a. Define the result categories to be produced by the assimilation action.
   b. Choose whether the assimilation action is abstract.
   c. Enter a short description in the Description field.
   d. Enter a more complete description in the Documentation field.

Table 9–1 (Cont.) Details Page

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result Categories</td>
<td>This multi-valued field lets you define the result categories for the assimilation action. A result category classifies the results of a scan. Different parts of a scan can be placed into different result categories.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Lets you add any other information about the action.</td>
</tr>
</tbody>
</table>
2. On the **UI Parameters** page, define the parameters the user enters in the Network Integrity GUI when running a scan.

3. On the **Model** page, define the parts of the static Information Model and the dynamic extension that the assimilation action uses to persist its data.

4. On the **Processors** page, define the flow of processors that comprise the assimilation action.
   a. If conditional logic is required, use the **Conditions** tab to define conditions for the processors. Use the **Processor** tab to associate the conditions with the processors.
   b. Define iterating logic using For Each constructs.
      See "About For Each Processors" for more information about the For Each construct.

To complete the editing of the assimilation action, the assimilation processors must be edited.

---

**Note:** Design Studio does not force any particular order for defining the editing. It is an iterative process in which you switch between various GUI screens to build up the model.

---

### Creating Assimilation Processors

There are several ways to create an assimilation processor. However, the best way is to create it using the Processor tab for the assimilation action. This method automatically adds the processor to the list of processors that the action uses, and it also ensures that only the supported type of processor is created for the current action.

**To create an assimilation processor:**

1. From the **Processor** tab of the assimilation action, in the action processors section, click **Add**.
   The Assimilation Processor Wizard appears.

2. Do not change the value in the **Project** field, as this field should already be set to the correct cartridge.

3. In the **Name** field, enter a name for the new processor.

4. In the **Folder** field, select a location for the processor. If you leave the **Folder** field blank, the action is created in the default location. The folders can be found inside the Assimilation Processors folder of the project tree in the Cartridge view.

5. Click **Finish**.
   Studio creates the new processor and displays it in the action editor. The Details page is displayed.

6. Click **Save** to save the assimilation action.

7. Double-click the new assimilation processor in the Actions Processors list to open the new assimilation processor editor.

### Configuring Assimilation Processors

The assimilation processor editor contains several tabbed pages.
The following link takes you to reference topics for the pages used to define an assimilation processor:

- Details Page
- Context Parameters Page
- Properties Page

### Details Page

The Details page contains general settings for the processor, as shown in Table 9–2.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lets you provide a short description of the processor.</td>
</tr>
<tr>
<td>Implementation Class</td>
<td>Lets you specify the implementation class for the processor. Design Studio automatically generates an interface class which the implementation class must implement. You can select an existing class, or create a class by selecting the Implementation Class link. If you use the link, the implementation class is created to match the processor interface.</td>
</tr>
<tr>
<td>Implementation Prefix</td>
<td>You cannot change this field. It is set automatically based on the name of the import processor. This field's value is used by Design Studio when creating some deployable artifacts.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Lets you add any other information about the processor.</td>
</tr>
</tbody>
</table>

### Context Parameters Page

The Context Parameters page is used to define the processor's inputs and outputs.

See "About Context Parameters" for more information about the tasks done using the Context Parameters page.

### Properties Page

Use this page to define the properties for the assimilation processor. Properties are name value pairs that are passed to the processor.

See "About Properties and Property Groups" for more information about properties configuration.

### Editing Assimilation Processors

This section provides an outline of the main steps involved in editing an assimilation processor.

1. Define properties to be passed to the assimilation processor using the Properties page.
2. On the Context Parameters page, define the inputs and outputs for the assimilation processor.
3. Use the Details page to specify the implementation class.

When the editing of the assimilation action and processors is completed, complete the assimilation action by coding the implementations for the assimilation processors.
Implementing Assimilation Processors

Many deployment artifacts for the assimilation action and its processors are generated automatically while editing. However, you must supply implementations for the assimilation processors using the `invoke` method.

Two forms of this method are shown in the following code fragments:

```java
// Signature for processor which does not have output parameters
public void invoke(AssimilationProcessorContext context,
                   ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
}

// Signature for processor which has output parameters
public ExampleProcessorResponse invoke(AssimilationProcessorContext context,
                                        ExampleProcessorRequest request) throws ProcessorException {
    // TODO Auto-generated method stub
    return null;
}
```

The parameters and return type of the `invoke` method are:

- **Processor_NameProcessorResponse**: this is the return type, for processors that have output parameters. For processors that do not have output parameters, the return type is `void`. This class is generated by Design Studio. It is a value object containing values for each of the processor’s output parameters. For processors that have output parameters, the `invoke` method must create a `ProcessorResponse` object, set its values and return the `ProcessorResponse` object.

- **Processor_NameProcessorRequest**: this is a value object, which has the following getters:
  - If UI Parameters are specified for the assimilation action, there is a getter that returns a UI Parameters value object
  - If properties are defined for the assimilation processor, there is a getter that returns a `Processor_NameProcessorProperties` value object
  - There is a getter for each input parameter that is defined for the processor.

This class is generated by Design Studio.

- **AssimilationProcessorContext context**: this is an SDK type, which has the following methods:
  - `getActionName`: returns the name of the action under which the processor is executing
  - `getProcessorName`: returns the name of the processor
  - `persistResults`: causes POMS objects to be flushed to the database. This helps to reduce memory consumption. See "About Persist Results" for more information.
  - `addToResult`: this is used to add a graph of POMS objects to the database under a result group. This method takes three parameters:
    * `String resultGroupName`: this is the name of a result group under which the results are persisted
    * `String resultGroupType`: this is the type of the result group under which the results are persisted. This should match a category defined on the action.
* DiscrepancyEnabled result: this is the root of result object graph to be persisted.

- getLatestResultGroupsInScope: returns an Iterator DisResultGroup, which is the latest results in scope. This is essentially the discovery or assimilation scan inputs to the assimilation action.

- getLatestScanRunsInScope: returns an Iterator DisScanRun, which is the latest scan runs in scope.

  This is also essentially the discovery or assimilation scan inputs to the assimilation action but includes several other objects from the Network Integrity model.

  These additional Network Integrity model objects might be useful in carrying out assimilation processing in some cases.

- getPreviousAssimilationScanRun: returns the latest completed scan run for the current assimilation scan. Use this to look at previous results, comparing current scope with previous scope.

- haveAllLatestScansInScopeChanged: returns true if any of the following conditions are met; false otherwise:
  * This is the first scan run for the assimilation scan
  * The latest scan run of every scan that is in the scope of both the previous assimilation run and the current assimilation run is more recent than the previous assimilation run

- haveLatestScanInScopeChanged: returns true if any of the following conditions are met; false otherwise:
  * This is the first scan run for the assimilation scan
  * At least one scan run in scope is more recent than latest assimilation scan run
  * The scope of the assimilation scan has changed between this run and the previous run

  This function is used to avoid unnecessary assimilation processing.

- getResultGroup: used to get an existing result group from your current scan if you need to access the graph of POMS objects previously added to a result group. This method takes two parameters:
  * String resultGroupName: this is the name of a result group under which the results are persisted.
  * String resultGroupType: this is the type of result group under which the results are persisted. This should match a category defined on the action.
This chapter provides an overview of discrepancies in Oracle Communications Network Integrity.

**About Discrepancies**

When Network Integrity detects a difference while comparing import and discovery data, it generates a discrepancy. The discrepancy captures all vital information about the difference, such as the entity and the name of the attribute or relationship containing the difference, the type of difference, and the values on both sides (that is to say, on the Compare, and the Reference sides).

These topics are further explored in:

- About the Compare and Reference Sides
- About Discrepancy Types
- About Discrepancy Status
- About Discrepancy Detail

**About the Compare and Reference Sides**

When dealing with discrepancies, the data from the two sides are named *Compare* and *Reference*. The significance is that the *Compare* side is the side of the scan that triggered the discrepancy comparison.

If a scan using a discovery action was also configured to detect discrepancies, the discrepancies created by that scan have discovery data on the *Compared* side, and import data on the *Reference* side.

On the other hand, if a scan uses an import action with detect discrepancies configured, the *Compared* fields of a discrepancy contain import data, and the *Reference* fields contain discovery data.

The discrepancy field *CompareSource* holds a value that indicates the origin of the compare-side data. The value is NETWORK for a discovery or an Assimilation scan, or INVENTORY for an import scan.

Table 10–1 shows *CompareSource* values for different discrepancy origins.

<table>
<thead>
<tr>
<th>Discrepancy Origin</th>
<th>Compared Side</th>
<th>CompareSource</th>
<th>Reference Side</th>
<th>ReferenceSource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery Scan</td>
<td>Discovery Data</td>
<td>NETWORK</td>
<td>Import Data</td>
<td>INVENTORY</td>
</tr>
</tbody>
</table>
About Discrepancy Types

There are seven types of discrepancy; they can be divided into four groups of related issues.

- Attribute Value Mismatch. See "Attribute Value Mismatch".
- Extra Entity, Missing Entity. See "Extra Entity and Missing Entity".
- Extra Association, Missing Association. See "Extra Association and Missing Association".
- Ordering Error, Association Ordering Error. See "Ordering Error and Association Ordering Error".

Network Integrity does not allow new discrepancy types to be defined.

Attribute Value Mismatch

This discrepancy indicates that an entity exists in both the Compare and Reference results, but an attribute was found not to have the same value on both sides.

Each discrepancy reports a mismatch problem on a single attribute. An entity can have multiple Attribute Value Mismatch discrepancies reported, if it has several mismatched attributes on both sides.

Table 10–2 shows discrepancy attributes and descriptions.

### Table 10–2 Attribute Value Mismatch: List of Discrepancy Attributes

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareEntity</td>
<td>This is the target entity whose attribute has a mismatched value.</td>
</tr>
<tr>
<td>referenceEntity</td>
<td>This is the matching entity on the other side of the discrepancy detection.</td>
</tr>
<tr>
<td>childTargetEntity</td>
<td>Not used. This has no value.</td>
</tr>
<tr>
<td>attributeOrRelationshipName</td>
<td>This holds the name of the attribute containing the mismatch.</td>
</tr>
<tr>
<td>compareValue</td>
<td>The value of the attribute on the target entity.</td>
</tr>
<tr>
<td>referenceValue</td>
<td>The value of the attribute on the matching entity on the other side.</td>
</tr>
</tbody>
</table>

Extra Entity and Missing Entity

This discrepancy indicates that an entity (and any dependent children) is present on one side of the comparison, but is absent from the other side.

An Extra Entity discrepancy indicates that the entity is present in the Compared side, but not in the Reference side.

A Missing Entity discrepancy indicates the reverse: the entity is absent is the Compared side, but present in the Reference side.

Figure 10–1 shows a representation of the discrepancy types.
Introduction to Discrepancies

About Discrepancies

Figure 10–1  Examples of Extra Entity and Missing Entity

Table 10–3 shows discrepancy attributes and descriptions.

Table 10–3  Discrepancy Attributes and Descriptions

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareEntity</td>
<td>This is the parent entity on one side of the comparison.</td>
</tr>
<tr>
<td>referenceEntity</td>
<td>This is the parent entity on the other side of the comparison.</td>
</tr>
</tbody>
</table>
| childTargetEntity     | This is the extra child entity on one side.  
The entity exists on the Compared entity tree when the discrepancy type is Extra Entity.  
The entity exists on the Reference entity tree when the discrepancy type is Missing Entity. |
| attributeOrRelationshipName | This holds the name of the association on the parent entity, which references the childTargetEntity. |
| compareValue          | Not used. This has no value. |
| referenceValue        | Not used. This has no value. |

When resolving an Extra/Missing Entity discrepancy, the processor is tasked with either adding or removing an object from its target system. The processor must consider the system that it is managing (Import/Inventory or Discovery/Network), and examine the following discrepancy fields to determine the appropriate action:

- DiscrepancyType
- CompareSource

For example: A Discrepancy Resolution Processor is created to make corrections to an inventory system. When this processor receives an Extra Entity discrepancy, it must check the value of CompareSource. If this value is NETWORK, the extra entity occurs in the network, and therefore it must be missing from the inventory system. The processor takes the corrective action of creating this entity in the inventory system.

However, if the discrepancy type is still Extra Entity, and CompareSource value is INVENTORY, the extra entity occurs in inventory.
Table 10–4 shows the resolution operations for the example processor, given the actual factors to be considered. The Present in columns indicate the system has the extra entity. The Resolution Operation column lists the appropriate inventory operation to resolve this discrepancy.

**Table 10–4  Appropriate Resolution Operations for Sample Processor**

<table>
<thead>
<tr>
<th>Discrepancy Type</th>
<th>Compare Source</th>
<th>Reference Source</th>
<th>Present in Network</th>
<th>Present in Inventory</th>
<th>Resolution Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Entity</td>
<td>Network</td>
<td>Inventory</td>
<td>Yes</td>
<td>No</td>
<td>Add the network entity into Inventory.</td>
</tr>
<tr>
<td>Missing Entity</td>
<td>Network</td>
<td>Inventory</td>
<td>No</td>
<td>Yes</td>
<td>Remove the inventory entity.</td>
</tr>
</tbody>
</table>

**Note:** Table 10–4 assumes that the discrepancy detection action was triggered from a Discovery scan.

If the discrepancies are generated by a discrepancy detection action that listens for results from Import scans, the compare source and reference source are reversed, and subsequently, the appropriate inventory operations are reversed as well. (This situation is not usual, but is certainly possible.) See Table 10–5 for this example.

**Table 10–5  Appropriate Resolution Operations for Sample Processor (Import Scan)**

<table>
<thead>
<tr>
<th>Discrepancy Type</th>
<th>Compare Source</th>
<th>Reference Source</th>
<th>Present in Network</th>
<th>Present in Inventory</th>
<th>Resolution Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Entity</td>
<td>Inventory</td>
<td>Network</td>
<td>No</td>
<td>Yes</td>
<td>Remove the inventory entity.</td>
</tr>
<tr>
<td>Missing Entity</td>
<td>Inventory</td>
<td>Network</td>
<td>Yes</td>
<td>No</td>
<td>Add the network entity into Inventory.</td>
</tr>
</tbody>
</table>

Network Integrity does not report Missing Entity discrepancies on the circuit of a root entity when the root entity is absent from either the Compared side or the Reference side.

For example, if a discovery scan finds Device1 with circuits A and B in the network, and the same device exists in inventory, but with circuits A, B, and C, Network Integrity reports a Missing Entity discrepancy on circuit C in the network.

In the above example, Network Integrity can fully compare the results for Device1 from the Compared side and the Reference side.

However, by default, when Device1 is not listed in the discovery results, Network Integrity does not report Missing Entity discrepancies on the device.

You can build a discrepancy detection action or extend the base discrepancy detection action to report missing Entity discrepancies on root entities. See “Working with Discrepancy Detection Actions and Processors” for more information.

**Extra Association and Missing Association**

This discrepancy indicates that an association in one entity (source) referencing another entity (target) is present on one side of the comparison, but is absent from the other side.
An Extra Association discrepancy indicates that the association is present in the *Compared* side, but not in the *Reference* side.

A Missing Association discrepancy indicates the reverse: the association is absent in the *Compared* side, but is present in the *Reference* side.

Each discrepancy indicates a problem with a single direction of association. If two entities have a bidirectional association with each other, and this bidirectional association is completely missing on one side, two discrepancies are generated by Network Integrity. One discrepancy will have the source and target entities switched from the other one.

*Figure 10–2* shows a representation of the discrepancy types.

*Figure 10–2  Examples of Extra Association and Missing Association*

**Extra Association (Assoc+)**

**Missing Association (Assoc-)**

In this example, there is a Mapped Device Interface association from Port F2 to Interface iff/2 which is missing on one side.

*Table 10–6* shows discrepancy attributes and descriptions.

*Table 10–6  Extra Association and Missing Association: List of Discrepancy Attributes*

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareEntity</td>
<td>This is the source entity on one side of the comparison.</td>
</tr>
<tr>
<td>referenceEntity</td>
<td>This is the source entity on the other side of the comparison.</td>
</tr>
<tr>
<td>childTargetEntity</td>
<td>This is the target entity of the association.</td>
</tr>
<tr>
<td></td>
<td>The entity exists on the Compared side when the discrepancy type is Extra</td>
</tr>
<tr>
<td></td>
<td>Association.</td>
</tr>
<tr>
<td></td>
<td>It exists on the Reference side when the discrepancy type is Missing Entity.</td>
</tr>
<tr>
<td>attributeOrRelationshipName</td>
<td>This holds the name of the association on the source entity which references the childTargetEntity.</td>
</tr>
<tr>
<td>compareValue</td>
<td>Not used. This has no value.</td>
</tr>
<tr>
<td>referenceValue</td>
<td>Not used. This has no value.</td>
</tr>
</tbody>
</table>
The processor must examine the discrepancy to determine whether the appropriate resolution operation is to add the association, or to remove it.

Table 10–7 shows the appropriate operation, given the values of discrepancy type, compare source, and reference source within the discrepancy.

Table 10–7  Appropriate Resolution Operations for Sample Processor

<table>
<thead>
<tr>
<th>Discrepancy Type</th>
<th>Compare Source</th>
<th>Reference Source</th>
<th>Present in Network</th>
<th>Present in Inventory</th>
<th>Resolution Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Association</td>
<td>Network</td>
<td>Inventory</td>
<td>Yes</td>
<td>No</td>
<td>Add the association into the inventory entity.</td>
</tr>
<tr>
<td>Missing Association</td>
<td>Network</td>
<td>Inventory</td>
<td>No</td>
<td>Yes</td>
<td>Remove the association from the inventory entity.</td>
</tr>
</tbody>
</table>

If the discrepancies are generated by a discrepancy detection action that listens for results from Import scans, the compare source and reference source are reversed, and subsequently, the appropriate inventory operation are reversed as well. (This situation is not usual, but is certainly possible.)

Table 10–8 shows the appropriate operation for this particular situation.

Table 10–8  Appropriate Resolution Operations fo r Sample Processor: Result from Import Scan

<table>
<thead>
<tr>
<th>Discrepancy Type</th>
<th>Compare Source</th>
<th>Reference Source</th>
<th>Present in Network</th>
<th>Present in Inventory</th>
<th>Resolution Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Association</td>
<td>Inventory</td>
<td>Network</td>
<td>No</td>
<td>Yes</td>
<td>Remove the association from the inventory entity.</td>
</tr>
<tr>
<td>Missing Association</td>
<td>Inventory</td>
<td>Network</td>
<td>Yes</td>
<td>No</td>
<td>Add the association into the inventory entity.</td>
</tr>
</tbody>
</table>

Ordering Error and Association Ordering Error

In some cases, the ordering of child or associated entities is significant. This discrepancy indicates that matched entities appear in different orders between the two sides. The only difference between the two types of discrepancy is that an Ordering Error indicates a problem with a parent/child association, while an Association Ordering Error indicates a problem with some other association.

Table 10–9 shows discrepancy attributes and descriptions.

Table 10–9  Ordering Error and Association Ordering Error: List of Discrepancy Attributes

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareEntity</td>
<td>This is the source/parent entity on one side of the comparison.</td>
</tr>
<tr>
<td>referenceEntity</td>
<td>This is the source/parent entity on the other side of the comparison.</td>
</tr>
<tr>
<td>childTargetEntity</td>
<td>Not used. This has no value.</td>
</tr>
</tbody>
</table>
About Discrepancy Status

The discrepancy status field identifies the state of a discrepancy within its lifecycle. The following figure gives an overview of this lifecycle:

Figure 10–3 shows a representation of the discrepancy types.

Table 10–9 (Cont.) Ordering Error and Association Ordering Error: List of Discrepancy Attributes

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeOrRelationshipName</td>
<td>This holds the name of the association having the ordering problem.</td>
</tr>
<tr>
<td>compareValue</td>
<td>Not used. This has no value.</td>
</tr>
<tr>
<td>referenceValue</td>
<td>Not used. This has no value.</td>
</tr>
</tbody>
</table>

About Discrepancy Status

Every discrepancy begins with a status of OPENED when it is first detected. It can then be moved to one of two states by a user using a Web UI operation:

- IDENTIFIED, by using a resolution action menu item
- IGNORED, by using the Ignore menu item

When a discrepancy is in the IDENTIFIED state, a user can use the Submit operation to move it to the SUBMITTED state. At this point, the discrepancy has moved out of user control, and into the control of a resolution action.
The resolution action processes the submitted discrepancy, and reports the outcome by setting the status to:
- **PROCESSED**, or
- **FAILED**

If the status is **PROCESSED**, the operation has succeeded, and the discrepancy can no longer be acted upon. If the status is **FAILED**, it becomes available for the user to specify an operation again, just like when it was first opened.

A resolution action may set a discrepancy status to **RECEIVED** immediately after the submit operation. This status indicates that the resolution operation is in progress, and reports its final operation status later.

**About Discrepancy Detail**

Table 10–10 lists all the attributes of a discrepancy. The Java type of a discrepancy is `DisDiscrepancy`. Use Java getter and setter patterns to retrieve and set the attribute’s value; for example: `getPriority()` to get the value of priority, and `setPriority(String)` to change its value.

Although the setters for all attributes are public, most fields should not be directly set by the processors. The following fields are safe to be used by processor Java implementations:
- **priority**
- **notes**
- **discrepancyOwner**

The **status** and **failureReason** fields should be set using the context methods when inside a processor `invoke()` method. Otherwise, they can also be set using setters.

**Table 10–10 Discrepancy Attributes**

<table>
<thead>
<tr>
<th>DisDiscrepancy Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>DisDiscrepancyType (Enum)</td>
<td>The discrepancy type. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ATTRIBUTE_VALUE_MISMATCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EXTRA_ENTITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MISSING_ENTITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EXTRA_ASSOCIATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MISSING_ASSOCIATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ORDERING_ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ASSOCIATION_ORDERING_ERROR</td>
</tr>
<tr>
<td>severity</td>
<td>DisDiscrepancySeverity (Enum)</td>
<td>The severity of the discrepancy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The values are (from most severe to least):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- CRITICAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MAJOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MINOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- WARNING</td>
</tr>
<tr>
<td>entityName</td>
<td>String</td>
<td>The name of the entity for which this discrepancy is raised.</td>
</tr>
</tbody>
</table>
### Table 10–10 (Cont.) Discrepancy Attributes

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>externalEntityType</td>
<td>String</td>
<td>The name of the specification, if the entity has a specification. Otherwise, the same value as staticEntityType.</td>
</tr>
<tr>
<td>staticEntityType</td>
<td>String</td>
<td>The name of the base entity type of the entity.</td>
</tr>
<tr>
<td>attributeOrRelationshipName</td>
<td>String</td>
<td>This holds the name of the attribute or relationship having the discrepancy.</td>
</tr>
<tr>
<td>compareEntity</td>
<td>long (Weak Reference)</td>
<td>This is the entityID of the entity for which this discrepancy is raised.</td>
</tr>
<tr>
<td>compareSystem</td>
<td>DisSource (Enum)</td>
<td>Indicates whether the compare data comes from Network (Discovery) or Inventory (Import) system. Valid values are NETWORK and INVENTORY.</td>
</tr>
<tr>
<td>compareValue</td>
<td>String</td>
<td>This is used by attribute value mismatch discrepancies to hold the value of the attribute on the compare side.</td>
</tr>
<tr>
<td>compareSource</td>
<td>String</td>
<td>The source value of the compareEntity. This value is copied from the Source field of the Scan configuration used to discover/import this entity into Network Integrity.</td>
</tr>
<tr>
<td>referenceEntity</td>
<td>long (Weak Reference)</td>
<td>This is the entityID of the entity of the discrepancy on the opposite side to the compareEntity</td>
</tr>
<tr>
<td>referenceSystem</td>
<td>DisSource (Enum)</td>
<td>This indicates whether the reference data comes from Network (Discovery) or Inventory (Import) system. Valid values are NETWORK and INVENTORY.</td>
</tr>
<tr>
<td>referenceValue</td>
<td>String</td>
<td>This is used by attribute value mismatch discrepancies to hold the value of the attribute on the reference side.</td>
</tr>
<tr>
<td>referenceSource</td>
<td>String</td>
<td>This is the source value of the referenceEntity. This value is copied from the Source field of the Scan configuration used to discover/import this entity into Network Integrity.</td>
</tr>
<tr>
<td>childTargetEntity</td>
<td>long (Weak Reference)</td>
<td>Used by Extra/Missing discrepancies to indicate the child/target entityID of the entity of an association.</td>
</tr>
<tr>
<td>ancestorEntityName</td>
<td>String</td>
<td>This is the name of the ancestor (parent) entity for the discrepancy.</td>
</tr>
<tr>
<td>ancestorEntityType</td>
<td>String</td>
<td>This is the name of the specification, if the ancestor entity has a specification. Otherwise, it takes the same value as ancestorStaticEntityType.</td>
</tr>
<tr>
<td>ancestorStaticEntityType</td>
<td>String</td>
<td>This is the name of the base entity type of the ancestor entity.</td>
</tr>
<tr>
<td>parentResultGroup</td>
<td>DisResultGroup</td>
<td>This is a reference of the parent Scan Result Detail (that is, the Result Group) of the compareEntity.</td>
</tr>
<tr>
<td>path</td>
<td>String</td>
<td>This is the path to the entity for this discrepancy. It is a comma-delimited list of entityIDs that describes the path from the root entity. For Missing Entity and Missing Association discrepancies, it is the path to the compareEntity followed by the entityID of the referenceEntity. For other discrepancy types, it is the path to the compareEntity.</td>
</tr>
</tbody>
</table>
### Table 10–10 (Cont.) Discrepancy Attributes

<table>
<thead>
<tr>
<th>Discrepancy Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>String</td>
<td>This is a user-editable field used to indicate the priority of this discrepancy. This would typically be used for customer-specific categorization, enabling a finer control than using severity alone.</td>
</tr>
<tr>
<td>notes</td>
<td>String</td>
<td>This is a user-editable field for comments.</td>
</tr>
<tr>
<td>discrepancyOwner</td>
<td>String</td>
<td>This is a user-editable field used to indicate an external owner of the discrepancy. It may be used for other purposes if desired.</td>
</tr>
<tr>
<td>operation</td>
<td>String</td>
<td>This holds the name of the resolution action being invoked.</td>
</tr>
<tr>
<td>operationIdentifiedBy</td>
<td>String</td>
<td>This is the ID of the user who identified the resolution action (the UI action to set the resolution operation, before the submit operation).</td>
</tr>
<tr>
<td>operationSubmittedBy</td>
<td>String</td>
<td>This is the ID of the user who submitted the resolution action.</td>
</tr>
<tr>
<td>submittedTime</td>
<td>Date</td>
<td>This is the timestamp when the status changed to OPERATION_SUBMITTED.</td>
</tr>
<tr>
<td>status</td>
<td>DisDiscrepancyStatus(Enum)</td>
<td>This is the current status of this discrepancy. Valid values are: DISCREPANCY_OPENED, DISCREPANCY_IGNORED, OPERATION_IDENTIFIED, OPERATION_SUBMITTED, OPERATION_RECEIVED, OPERATION_NOT_IMPLEMENTED, OPERATION_PROCESSED, OPERATION_FAILED</td>
</tr>
<tr>
<td>lastStatusChangeTime</td>
<td>Date</td>
<td>This is the timestamp when the status attribute was last updated.</td>
</tr>
<tr>
<td>reasonForFailure</td>
<td>String</td>
<td>This holds the error message set by the processor using context.discrepancyFailed() method.</td>
</tr>
<tr>
<td>entityID</td>
<td>long</td>
<td>This is an Internal identifier.</td>
</tr>
</tbody>
</table>
This chapter provides information about Oracle Communications Network Integrity discrepancy detection actions and processors.

**Discrepancy Detection Action**

The discrepancy detection action is a Network Integrity operation that compares discovery and import scan results, and reports on their differences by generating discrepancies.

A detection action triggers to run immediately following a discovery, import, or assimilation scan run. (Select the Detect Discrepancy check box in the scan configuration to set the trigger.) The entity results from the triggering scan run become the Compare entities for the detection action. The action then uses a matching algorithm to find from the other side, and precedes with the comparisons.

See "About the Compare and Reference Sides" for a fuller description of the two sides of entities of discrepancy detection.

See "About the Base Detection Cartridge and the Default Comparison Algorithm" for a description of the comparison algorithm.

Create a discrepancy detection action whenever new discovery, import, or assimilation actions are created, because every detection action is configured to receive results from specific actions only. See "Result Source" for more information.

**About Discrepancy Detection**

Discrepancy detection triggers immediately after a scan is finished. A scan is configured to use a single type of action, and therefore only generates Discovery results (representing network entities) or Import results (representing inventory system entities). Therefore, when the discrepancy detection action triggers, it has immediate access to one side of results: the compare entities.

For the other side of the results, the detection action searches the Network Integrity database for results with the following criteria:

- The results must come from the opposite system from the triggered scan. For example, if the detection action triggers from a Discovery action scan, then the detection action searches the database for Import result.

- The results have a matching name and result category (as configured by Result Source).
The results must come from the most recent scan result.

If no matching results are found, then EXTRA_ENTITY discrepancies are generated for each root entity on that result.

Creating a Discrepancy Detection Action

See Design Studio Network Integrity online Help for information on creating actions.

Discrepancy Detection Action Editor

The Discrepancy Detection Action editor has the following five tabbed pages:

- **Details**: For discrepancy detection actions, you can edit the Description, Implementation Prefix, Abstract setting, and Documentation text. See the Common Fields for Details Tab section.

- **Processors**: The only processor type allowed to be created and used is the Discrepancy Detection Processor. See the Processors Tab and Common Fields for Processors Tab sections.

- **Model**: See the Model Collection and Common Fields for Model Tab sections for more detail.

- **Result Source**: Specifies the input result categories for the action to act upon.

- **Conditions**: See the Conditions and Common Fields for Conditions Tab sections for more detail.

Result Source

The Result Source table specifies a list of scan actions that can trigger a Discrepancy Detection Action. The triggering action must be a discovery, import, or assimilation action. By default, results from all categories are included in the discrepancy detection. It is possible to choose a subset of the categories to apply the discrepancy detection.

For example, Figure 11–1 illustrates a Cisco router discovery action that produces results in 2 categories: **Device** and **VPN**. Two separate detection actions are written to compare the results. Each detection action specifies a result source with the same Action, but different result category. For example, the Device Discrepancy Detection Action receives results of Device category only.
Figure 11–1  Discrepancy Detection Actions (Example 1)

A Result Source that does not specify a Result Category matches every Result Category generated by the scan Action. Figure 11–2 illustrates a Cisco Discrepancy Detection Action that receives both Device and VPN categories of results.

Figure 11–2  Discrepancy Detection Action (Example 2)

The Result Source is a mandatory field; there must be at least one entry in the table. Design Studio marks the Discrepancy Detection Action with an error during a project build if the table has no entries.

Caution: No two discrepancy detection actions can have the same Label and Result Source.

About Result Source and Scan Types

Typically a Result Source configuration detection action has a single action as the result source: usually the Discovery action. This detection action triggers when an Integrity Scan is configured using that exact Discovery action, and the Detect Discrepancy option is checked. This detection action does not trigger by scans configured with any other Discovery or Import action. Do not set the Detect
**Discrepancy** option on the Import scan, because this might not trigger a detection action at all.

### Generated Action MDB and Controller

The Detection Action is implemented as an MDB. Please see the Generated Action MDB and Controller subsection of the General Introduction to Actions section for more information.

### Discrepancy Detection Processor

The Discrepancy Detection Processor is the atomic sub-function of a Discrepancy Detection Action. The typical tasks of a detection processor are different than the scan-related Processors (Discovery, Import, and Assimilation) and include the following:

- create and add filters to alter the default behavior of the base Discrepancy Detection Action
- perform post-processing on the set of discrepancies produced by the base Discrepancy Detection Action

See the section "Discrepancy Detection Processor Patterns" for more information on the various patterns for detection action-processor implementation.

### Creating a Discrepancy Detection Processor

See Design Studio Network Integrity online Help for information on creating processors.

### Implementation Interface

The processor implementation class derives from a Studio-generated interface class. There is a single abstract method that the implementation class must implement. The abstract method has the following interface:

```java
public <Processor_Name>Response invoke(
    DiscrepancyDetectionProcessorContext context,
    <Processor_Name>Request request)
    throws ProcessorException
{
    // Implementation...
}
```

#### Input Parameters of Invoke()

Table 11–1 shows the first parameter, context, that provides the following methods:

<table>
<thead>
<tr>
<th>Context Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getActionName()</td>
<td>String</td>
<td>Getter for the name of this Action.</td>
</tr>
<tr>
<td>addDiscrepancy(DisDiscrepancy discrepancy)</td>
<td>void</td>
<td>Adds a new discrepancy into the result set.</td>
</tr>
<tr>
<td>getDiscrepancies()</td>
<td>Collection&lt;DisDiscrepancy&gt;</td>
<td>Returns the set of result discrepancies added so far.</td>
</tr>
</tbody>
</table>
In addition, the context can be cast into class `BaseDiscrepancyDetectionController` (package oracle.communications.integrity.scanCartridges.sdk) that makes the method in Table 11–2 available.

<table>
<thead>
<tr>
<th>Context Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>persistDiscrepancies()</td>
<td>void</td>
<td>Writes the set of result discrepancies into the database.</td>
</tr>
<tr>
<td>findReferenceRoot(</td>
<td>DiscrepancyEnabled</td>
<td>Returns the referenceEntity that matches all criteria of the input compareEntity.</td>
</tr>
<tr>
<td></td>
<td>DiscrepancyEnabled compareEntity)</td>
<td></td>
</tr>
<tr>
<td>getRootEntityLoader()</td>
<td>RootEntityLoader</td>
<td>(For advanced use only.) Returns the root entity loader that gets all the root entities on the result group on the compare side. Root entities are the starting entities of the comparison process.</td>
</tr>
<tr>
<td>getDiscrepancyDetector()</td>
<td>GenericDiscrepancyDetector</td>
<td>(For advanced use only.) Returns the discrepancy detector object that starts off the comparison process.</td>
</tr>
<tr>
<td>isCompareRoot(</td>
<td>boolean</td>
<td>(For advanced use only.) Returns true if the input entity is a Compare Root Entity. Root entities are the starting entities of the comparison process.</td>
</tr>
<tr>
<td></td>
<td>DiscrepancyEnabled compareEntity)</td>
<td></td>
</tr>
</tbody>
</table>

The request parameter contains getters for each item in the Input Parameters table of the Context Parameters tab of the processor editor. It also contains a getter to retrieve the groups and items listed in the Properties tab.

**Return Type of Invoke()**

The return type of the invoke() method varies depending on the Output Parameters setting in the Context Parameters tab page.

If there is no output parameter, the return type is void.

If there are one or more output parameters, the return type is a generated class with the name `<Processor_Name>Response`. This Response class has getters and setters for each item in the Output Parameters table of the Context Parameters tab of the processor editor.

**Discrepancy Detection Processor Patterns**

There are several patterns of processor used inside a discrepancy detection action. Each successive pattern introduces a new level of flexibility, power, and complexity. The patterns are listed below, in order from the simplest to the most complex:

1. Reusing the base Detect Discrepancy Action.
2. Adding new filters and handlers.
3. Adding post-Processors.

Reusing the Base Detect Discrepancy Action

This usage pattern provides a baseline comparison algorithm between the compare and the reference sides. A detection action using this pattern has the ability to compare exact entity attributes and associations, and can generate five of the seven types of discrepancy. (Ordering Errors and Association Ordering Errors are not detected by the baseline comparison algorithm, because it assumes that there are no ordered relationships.)

To use this pattern, use following steps:

1. Import the Base Detection Cartridge into the Studio Project environment. This cartridge contains a reusable detection action called **Detect Discrepancies**.
2. Create a Discrepancy Detection Action.
3. In the **Processors** tab, add the **Detect Discrepancies** action. This action adds a single processor, **Discrepancy Detector**, into the table.
4. Set the **Result Source** for this action.

**Figure 11–3** shows the content of the processors tab for a detection action using this pattern.

**Figure 11–3 Processor Tab Detection Action**
About the Base Detection Cartridge and the Default Comparison Algorithm

The Base Detection cartridge contains a reusable Discrepancy Detection Action called **Detect Discrepancies**. This action is abstract, and cannot be deployed by itself. It is intended to be imported by virtually all other Discrepancy Detection Actions. This action implements a general comparison algorithm that can work with all entity types and specifications, and can detect and report all seven types of discrepancy.

This ability enables a cartridge developer to build a working discrepancy detection cartridge for arbitrary discovered data without writing code. Its behavior is customizable, by using the techniques described in the following processor patterns.

The default comparison algorithm is outlined below.

1. The detector loops over the compare root entities.

2. The detector checks if each compare root entity should be considered for discrepancy detection. If it should not, the root entity is ignored, and the detector begins processing the next compare.

3. A rootEntityHandler finds the matching reference root entity for the compare root entity. The default rootEntityLoader uses the Name field to find the matching reference root entity. If no reference root entity is found, an EXTRA_ENTITY discrepancy is generated.

4. The attributes of the matching entities are compared, and an ATTRIBUTE_VALUE_MISMATCH discrepancy is generated for each attribute with different values. If an attribute contains an ordered list of values, an ORDERING_ERROR discrepancy is generated if the order of the values does not match.

5. The associations of the matching entities are compared, and an EXTRA_ASSOCIATION or MISSING_ASSOCIATION discrepancy is generated for unmatched target entities of an association. The default relationship handler uses the Name field to match related entities of the compare and reference sides. If an association is an ordered association, an ASSOCIATION_ORDERING_ERROR discrepancy is generated if the order of the matching associated entities is different.

6. The child relationship of the matching entities is compared, and an EXTRA_ENTITY or MISSING_ENTITY discrepancy is generated for unmatched child entities. The default relationship handler uses the Name field to match child entities of the compare and reference sides. If a child relationship is an ordered association, then an ORDERING_ERROR discrepancy is generated if the order of the matching child entities is different.

7. The comparison continues by applying the above algorithm to all children entities recursively, until all entities have been checked. The comparison also stops at a given entity if one of the following is true: the entity is a compare root entity, or the entity is flagged as a shadow entity.

The Detect Discrepancy action creates discrepancies with a default severity of CRITICAL for EXTRA_ENTITY and MISSING_ENTITY, and WARNING for the other types.

Adding New Filters and Handlers

This usage pattern builds on the Reuse pattern by adding filters and handlers to customize the general comparison algorithm. The following changes can be achieved:

- which root discovery entities are of interest
how to match discovery entities to import entities
which attributes are not significant for a particular entity type
how to compare a particular attribute
which relationships to consider for a particular entity type
what severity to apply to a discrepancy
define a relationship as ordered (to automatically add ORDERING checks)
set a default/suggested resolution action (e.g. Ignore, Correct in UIM, etc.)

To use this pattern, follow the Reuse pattern to create your detection action, and then create one new detection processor, and move it above the Discrepancy Detector processor in the table. This new processor becomes the Filter Initializer processor for the detection action. (For example, in Figure 11–4, a new action follows this pattern by having its own Sample Filter Initializer processor placed above the imported Discrepancy Detector processor.)

Figure 11–4  Sample Filter Initializer

The main task of a Filter Initializer Processor is to register filters and handlers for use by the subsequent Discrepancy Detector processor. Handlers are code that implements various behaviors used during discrepancy detection. Filters are code that manipulates the handlers to be used by discrepancy detection.

About Filters

There are four different types of filters that can be added by the processor:
- **AttributeFilter**: This filter is called during the assignment of attribute handlers for the given entity type. This filter can add, modify and remove handlers from the given `attributeHandlers`.

- **RelationshipFilter**: This filter is called during the assignment of relationship handlers for a given entity type. This filter can add, modify and remove handlers from the given `relationshipHandlers`.

- **DiscrepancyFilter**: This filter is called during assignment of discrepancy handlers for a given entity type. This filter can modify or remove the default `discrepancyHandler`.

- **RootEntityFilter**: This filter is called during the assignment of the root entity handler for a given entity type. This filter can replace the default `rootEntityHandler` with another one.

### About Handlers

There are four different types of handlers that can be manipulated by their associated filters:

- **AttributeHandler**: This handler can change the mapping of attributes, or change the behavior of the comparison operation. For example, a string comparison is normally case-sensitive. An `AttributeHandler` can be added to cause a case-insensitive comparison to be used instead.

  Network Integrity provides a `DefaultAttributeHandler` class which implements the necessary `AttributeHandler` interface and the default case-sensitive string comparison behavior. To override this behavior, create a class which subclasses `DefaultAttributeHandler`, and then override the following method:

  ```java
  protected boolean equalsNonNull(Object a1, Object a2);
  ```

- **RelationshipHandler**: This handler can change the mapping of relationships. For example, a relationship comparison would normally check the identically-named relationship on the reference entity. A `RelationshipHandler` can be added which causes a differently-named relationship to be used instead.

  Network Integrity provides a `DefaultRelationshipHandler` class that implements the necessary `RelationshipHandler` interface, and has knowledge of all relationships for each supported Oracle Communications Information Model entity type. The following method can be overridden by a new subclass to alter the default behavior:

  ```java
  protected Object getKey(DiscrepancyEnabled entity)
  ```

  This method gets a key value that distinguishes a single entity from a set of entities within a single relationship. The `DefaultRelationshipHandler` implementation returns the value of the `Name` attribute for the input entity.

- **DiscrepancyHandler**: This handler can change the fields of a discrepancy immediately after it is generated. It can also completely remove the discrepancy. An example of its use is to adjust the severity value of a discrepancy of a DeviceInterface entity based on its Speed value.

  Network Integrity provides a non-accessible default `DiscrepancyHandler` implementation which does nothing. To override this behavior, create a class which implements the `DefaultHandler` interface, and implement the following method:

  ```java
  DisDiscrepancy processDiscrepancy(DiscrepancyEnabled currentEntity,
  ```
The overridden method should alter the input `generatedDiscrepancy`, and then return it.

- **RootEntityHandler**: This handler changes the algorithm for finding a matching reference entity for an input compare entity. An example of its use is to change the default comparison criteria to using the ID field to find the match, instead of the default of using Name field.

  See Example 4: Using Root Entity Filter and Handler for a full example of the proper setup and usage of a root entity handler.

### Filters and CimType

Filters register against one or more types of Information Model entities produced by a Discovery, Import, or Assimilation scan. Filters can also register against one of more Specifications of an entity type, for more fine-grained control.

In Java code, the entity type and specification are designated by using the class CimType. To register a filter against an entity type (for example, Equipment), use the single parameter constructor for CimType:

```java
CimType eqType = new CimType(Equipment.class);
```

To register a filter against a particular specification (for example, `cevSensorClock`, an Equipment specification defined in the Cisco UIM cartrige), use the two-parameter constructor for CimType:

```java
CimType clockEqType = new CimType(Equipment.class, "cevSensorClock");
```

It is possible to take advantage of the inheritance model of the Information Model entity classes to register quickly against several classes with one call. For example, all Information Model entities that support discrepancy detection inherit from the class `DiscrepancyEnabled`. Therefore, the following code CimType can register a filter against everything:

```java
CimType allType = new CimType(DiscrepancyEnabled.class);
```

### Filter and Handler Examples

The next four examples demonstrate the four types of filters and handlers. The prerequisite tasks for all examples are to:

1. Create a Discrepancy Detection Action.
2. Set the Result Source in the Result Source tab.
3. Add the Detect Discrepancy action in the Processors tab.
4. Create a filter initializer Processor.
5. Move the new processor above the Discrepancy Detector processor.

**Example 1a: Using Attribute Filter and Handler (Static Attribute)**

The following code fragments shows how to add an attribute filter to ignore the static attribute `description` on LogicalDevices. The result of this code is that the new detection action does not generate any description Attribute Value Change discrepancies on LogicalDevices.
1. Define the filter class and remove the handler for the attribute **description**.

```java
private class LogicalDeviceAttributeFilter implements AttributeFilter {
    public void filterAttributes(CimType cimType,
        Map<String, AttributeHandler> attributeHandlers) {
        attributeHandlers.remove("description");
    }
}
```

2. In the `processor invoke()` method, get the generic discrepancy detector from the context.

```java
GenericDiscrepancyDetector detector = context.getDiscrepancyDetector();
```

3. In the `invoke()` method, create the CIMType object to name the entity type, and add the custom filter.

```java
CimType ldType = new CimType(LogicalDevice.class);
detector.addFilter(ldType, new LogicalDeviceAttributeFilter());
```

**Example 1b: Using Attribute Filter and Handler (Characteristic)**

The following code fragments show how to add an attribute filter to ignore the characteristic `systemObjectId` on LogicalDevice entities with the specification `DemoLogicalDevice`. The main difference between this example and the previous example is Step 3, where the Specification name must be included in the CimType constructor.

1. Define the filter class and remove the handler for the attribute **systemObjectId**.

```java
private class DemoLogicalDeviceAttributeFilter implements AttributeFilter {
    public void filterAttributes(CimType cimType,
        Map<String, AttributeHandler> attributeHandlers) {
        attributeHandlers.remove("systemObjectId");
    }
}
```

2. In the `processor invoke()` method, get the generic discrepancy detector from the context.

```java
GenericDiscrepancyDetector detector = context.getDiscrepancyDetector();
```

3. In the `invoke()` method, create the CIMType object to name the entity type and the specification, and add the custom filter.

```java
CimType ldType = new CimType(LogicalDevice.class, "DemoLogicalDevice");
detector.addFilter(ldType, new DemoLogicalDeviceAttributeFilter());
```

**Example 2: Using Relationship Filter and Handler**

In this example, the detection action to skips the `physicalPorts` relationship of all Equipment entities. By using the following code fragment, the new detection action no longer examines any children ports of equipment.

1. Define the filter class and remove the relationship handler for the relationship `physicalPorts`.

```java
private class EquipmentRelationshipFilter implements RelationshipFilter {
    public void filterRelationships(CimType cimType,
        Map<String, AttributeHandler> attributeHandlers) {
```

2. In the `processor invoke()` method, get the generic discrepancy detector from the context.

```java
GenericDiscrepancyDetector detector = context.getDiscrepancyDetector();
```

3. In the `invoke()` method, create the CIMType object to name the entity type and the specification, and add the custom filter.

```java
CimType ldType = new CimType(Equipment.class, "DemoLogicalDevice");
detector.addFilter(ldType, new EquipmentRelationshipFilter());
```
2. In the processor `invoke()` method, get the generic discrepancy detector from the context.

```java
genericDiscrepancyDetector detector = context.getDiscrepancyDetector();
```

3. In the `invoke()` method, create the CIMType object to name the entity type, and add the custom filter.

```java
cimType eqType = new CimType(Equipment.class);
detector.addFilter(eqType, new EquipmentRelationshipFilter());
```

### Example 3: Using Discrepancy Filter and Handler

This example sets the severity to Minor on every Missing Entity and Extra Entity discrepancy generated by the new detection action. Use the following code fragment for this task:

1. Define the filter class and add a new discrepancy handler. This handler performs a discrepancy type check, and sets the severity accordingly.

```java
private class CustomDiscrepancyFilter implements DiscrepancyFilter {
    public DiscrepancyHandler filterDiscrepancies(
        CimType cimType,
        DiscrepancyHandler handler) {
        return new DiscrepancyHandler() {
            public Discrepancy processDiscrepancy(
                DiscrepancyEnabled cimBase,
                Discrepancy discrepancy) {
                if (DiscrepancyType.EXTRA_ENTITY ==
                    discrepancy.getType() ||
                    DiscrepancyType.MISSING_ENTITY ==
                    discrepancy.getType()) {
                    discrepancy.setSeverity(DiscrepancySeverity.MINOR);
                }
                return discrepency;
            }
        };
    }
}
```

2. In the processor `invoke()` method, get the generic discrepancy detector from the context.

```java
genericDiscrepancyDetector detector = context.getDiscrepancyDetector();
```

3. In the same `invoke()` method, create the CIMType object to name the entity type, and add the custom filter.

```java
cimType allType = new CimType(DiscrepancyEnabled.class);
detector.addFilter(allType, new CustomDiscrepancyFilter());
```
Example 4: Using Root Entity Filter and Handler

This advanced technique in this example changes the matching algorithm that finds the matching reference entity for any compare entity. The default algorithm finds matches based on a comparison of the value of the **name** attribute. This example changes the comparison to use the **nativeEmsName** attribute instead.

---

**Note:** This feature is used in the MIB II UIM cartride.

---

The example is in two parts. The first part alters the root entity handler to match compare root entities with reference root entities using the **nativeEmsName** attribute. The second part use relationship handlers to make the discrepancy detector use **nativeEmsName** attribute to distinguish the children.

First, the root entity filter and handler code fragments are as follows:

1. Define a method in the new processor to create the root entity filter. This filter creates a new root entity handler and returns it.

   ```java
   private RootEntityFilter getRootEntityFilter() {
     return new RootEntityFilter() {
       @Override
       public RootEntityHandler filterRootEntities(CimType arg0, RootEntityHandler arg1) {
         return new MatchRootEntityByNativeEmsNameInsteadOfName();
       }
     };
   }
   ```

2. Define a private class that extends from **DefaultRootEntityHandler**. This class is the one created in step 1. Override the **getReferenceRootEntity()** method as follows. Notice the use of a string array containing the string **nativeEmsName** to specify the use of this attribute. Also notice the use of a **RuntimeException** to report problems.

   ```java
   private class MatchRootEntityByNativeEmsNameInsteadOfName extends DefaultRootEntityHandler {
     @Override
     public DiscrepancyEnabled getReferenceRootEntity(DiscrepancyEnabled compareRoot) {
       try {
         PomsManagerFactory factory = new PomsManagerFactory();
         DisResultGroupManager DisResultGroupManager = factory.getDisResultGroupManager();
         DisResultGroup g = DisResultGroupManager.getDisResultGroup(
           (Persistent) compareRoot);
         return new ReferenceRootFinder(g).
           findReferenceRoot((Persistent) compareRoot, new String[] { "nativeEmsName" });
       } catch (Exception e) {
         logger.log(Level.SEVERE,
           "Error while getting reference root, compareRoot " +
           compareRoot, e);
         throw new RuntimeException("Error while getting reference root, Aborting discrepancy
           generation",
           e);
       }
     }
   }
   ```
3. In the `invoke()` method of the processor, create the CIMType object to cover all entity types, and add the root entity filter defined in step 1.

   ```java
   CimType allType = new CimType(DiscrepancyEnabled.class);
   context.getRootEntityLoader().addFilter(allType, getRootEntityFilter());
   ```

   Part two adds a relationship filter to each entity type that the detection processor expects to encounter. This code fragment example shows a change to a single entity type. It changes the LogicalDevice to DeviceInterface child relationship to match using `nativeEmsName` instead of `name`. Normally, this code pattern needs to be repeated once for each entity type. (See the MIB II UIM and Cisco UIM cartridge packs for a full example.)

1. Define the relationship handler as a class inside the processor’s class. This class should inherit from `DefaultRelationshipHandler`, and override the `getKey()` method to return

   ```java
   public class MatchDevIntfByNativeEmsName extends DefaultRelationshipHandler {
       @Override
       protected Object getKey(DiscrepancyEnabled entity) {
           return ((DeviceInterface) entity).getNativeEmsName();
       }
   }
   ```

2. In the processor `invoke()` method, get the generic discrepancy detector from the context.

   ```java
   GenericDiscrepancyDetector detector = context.getDiscrepancyDetector();
   ```

3. In the same `invoke()` method, create the CIMType object to name the entity type, and add the custom filter.

   ```java
   CimType ldType = new CimType(LogicalDevice.class);
   detector.addFilter(ldType, new RelationshipFilter() {
       @Override
       public void filterRelationships(
           CimType cimType,
           Map<String, RelationshipHandler> relationshipHandlers) {
           relationshipHandlers.put("deviceInterface",
               new MatchDevIntfByNativeEmsName());
       } // end filterRelationships
   }); // end new RelationshipFilter
   ```

Adding Post-Processors

This usage pattern builds on the Reuse pattern and adds processors after the Discrepancy Detector processor. These post-processors access the full set of detected discrepancies using the `getDiscrepancies()` method of the DiscrepancyDetectionProcessorContext object (context). Because they are not persisted until all processors in the action have executed, the discrepancies can be manipulated completely by the post-processors. They can be modified or removed. Also, new discrepancies can be added.

Although all fields of a discrepancy can be modified by using setters, there are many fields that should not be altered. The following discrepancy fields can be safely changed by post-processors:

- priority, notes, discrepancyOwner
- severity, compareValue, referenceValue
- operation + operationIdentifiedBy + status (status set to OPERATION_IDENTIFIED) (Must be set together.)

Any other discrepancy fields should not be altered; otherwise, discrepancy resolution actions may suffer errors and failures.

An example of the use of post-processors is to automatically assign all CRITICAL severity discrepancies to a specific department (using the discrepancyOwner field). The following code snippet from a post-processor shows how this is done.

```java
@override
define void invoke(DiscrepancyDetectionProcessorContext context,
                        DiscrepancyPostProcessorProcessorRequest request)
  throws ProcessorException {
  for (DisDiscrepancy discrepancy : context.getDiscrepancies()) {
    if (discrepancy.getSeverity().equals(DisDiscrepancySeverity.CRITICAL)) {
      discrepancy.setDiscrepancyOwner("Sherlock Holmes");
    }
  }
}
```
A discrepancy resolution action is an extendable Network Integrity operation which acts on an external system to resolve a discrepancy. For example, a resolution action updates a mismatch in an inventory system using information gathered from the network or generates a trouble ticket to kick off a network configuration change process.

A discrepancy resolution action operation is initiated by a user of the Network Integrity user interface on the Manage Discrepancy page, using the following steps:

1. The user identifies the desired resolution action on selected discrepancies. Each discrepancy can have only one resolution action set.
2. The user submits the discrepancies with identified resolution actions to the system.

On receiving the submitted discrepancies, Network Integrity groups them based on their scan origin, result category, and resolution label, and then invokes the appropriate Discrepancy Resolution Action.

The Action then examines each discrepancy in detail, using the contained information to figure out the appropriate steps to resolve the problem.

As with other types of actions, a discrepancy resolution action is made up of a sequence of discrepancy resolution processors. The processors are shown in the Processor table in Design Studio. At the beginning of an action operation, these processors are invoked serially from top of the table to bottom. The first processor is given the list of submitted discrepancies marked for this action. This processor determines a subset of these discrepancies to handle (which can range from none to all), performs the resolution operation, and sets their status to Processed or Failed. Then, the next processor is given the remaining unhandled discrepancies for processing, and so on.

The action is complete when all the processors are invoked. If there are any discrepancies which remain unhandled at the end, their status is automatically set to Not Implemented.

The following sections in this chapter describe general information about implementing a resolution action. For a detailed discussion of a working sample, see the following documents included with the cartridges:

- Network Integrity Cisco Router and Switch UIM Integration Cartridge Guide
- Network Integrity MIB-II UIM Integration Cartridge Guide
Creating Discrepancy Resolution Actions

To create a Discrepancy Resolution action, see the Network Integrity Studio Help.

Discrepancy Resolution Action Editor

The Discrepancy Resolution action editor contains several tabbed pages.

The following link takes you to reference topics for the pages used to define an action:

- Details Page
- Processors Page
- Model Page
- Result Source Page
- Conditions Page

Details Page

The Details page contains general setting for the action.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution Action Label</td>
<td>See &quot;About the Resolution Action Label&quot;.</td>
</tr>
<tr>
<td>Description</td>
<td>Lets you provide a short description of the action.</td>
</tr>
<tr>
<td>Implementation Prefix</td>
<td>This takes the value of the component name by default. This field cannot be changed. This field is used by Design Studio when creating some deployable artifacts.</td>
</tr>
<tr>
<td>Abstract</td>
<td>Lets you specify whether the action is abstract.</td>
</tr>
<tr>
<td></td>
<td>If the action is abstract, then it can only be used as a component within another action.</td>
</tr>
<tr>
<td></td>
<td>If the action is not abstract, then it can be deployed and a scan can be executed from it.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Lets you add any other information about the action.</td>
</tr>
</tbody>
</table>

Processors Page

The Processors page is used to define the processors that comprise the action.

See "About the Processors Tab" for further information about the tasks carried out using the Processors tab.

The only processor type that can be created and used for an import action is the discrepancy resolution processor.

Model Page

Use the Model page to specify the model collections that the discrepancy resolution action uses. The model collection defines the parts of Oracle Communications Information Model and the dynamic extensions that are related to the model collection. In effect, the Model tab specifies which parts of the Information Model are used by the discrepancy resolution action.

See "About Model Collections in Actions" for further information about model collections.
Result Source Page

Specifies the input result categories for the action to act upon. See "About Result Sources" for further information.

Conditions Page

The Conditions page allows conditions to be defined. Conditions can be associated with processors to cause conditional execution of the processor.

See "About Conditions" for further information about conditions.

About the Resolution Action Label

The Resolution Action Label identifies the Discrepancy Resolution action in the Network Integrity User Interface (UI). It is displayed as a command in the Actions menu of the Discrepancy Search Results table of the Review Discrepancies page. Figure 12–1 displays the label corresponding to the command.

Figure 12–1 Resolution Action Label in Actions Menu of Network Integrity UI

This label is a mandatory field. Design Studio reports an error if this label has no value. The use of a command phrase as the label string is recommended. Some example labels are:

- Correct in Inventory System, or
- Open a Trouble Ticket

The label input field allows you to choose either a label from another Discrepancy Resolution action defined within your workspace, or to type in a new label. A label can be shared by multiple actions; this implies that multiple actions are sharing a single menu item in the Actions menu of the Discrepancies page.

Network Integrity determines the correct Action to invoke based on a combination of the label and the Result Source.

WARNING: No two discrepancy resolution actions can have the same Label and the same Result Source.
About Result Sources

The result source is a list of discrepancy filtering criteria. Each criterion represents a single source of discrepancy, and is specified by a combination of the originating scan action and a result category. A resolution action only receives discrepancies from the specified result categories which were created by scans using the specified actions.

Figure 12–2 shows an example of result sources being applied in Network Integrity.

A criterion that does not specify any result category matches all result categories generated by the scan action in the criterion.

Figure 12–3 shows a representation of the discrepancy types.

The result source is a mandatory field; there must be at least one entry in the table. Design Studio marks a discrepancy resolution action with an error during a project build if this table has no entries.

WARNING: No two discrepancy resolution actions can have the same Label and the same Result Source.
About Discrepancy Resolution Processors

The discrepancy resolution action is implemented as an MDB, just like any other Network Integrity action. See "Generated Action MDB and Controller" for more information.

Generated Action MDB and Controller

The discrepancy resolution action is implemented as an MDB, just like any other Network Integrity action. See "Generated Action MDB and Controller" for more information.

About Discrepancy Resolution Processors

The only type of processor available to the discrepancy resolution action is the discrepancy resolution processor.

As with other types of actions, the list of processors are invoked serially from top of the table to bottom. The first processor is given the list of submitted discrepancies marked for this action. This processor determines a subset of these discrepancies to handle (which can range from none to all), perform the resolution operation, and set their status to either Processed or Failed.

Then, the next processor is given the remaining unhandled discrepancies for processing, and so on. The action is complete when all the processors are invoked. If there are any discrepancies which remain unhandled at the end, their status is set to Not Implemented.

The discrepancy resolution processor is the Java implementation of a discrepancy resolution action. The processor carries out the following tasks:

- Filter through its input list of discrepancies to process only those discrepancies it can handle
- Communicate with the discovery or import system to correct a discrepancy
- Report the status of a correction operation

See "About Processor Implementation" for more information.

Creating a Discrepancy Resolution Processor

See Network Integrity Studio online Help for information about creating a discrepancy resolution processor.

Implementing a Discrepancy Resolution Processor

This section provides details about the discrepancy resolution processor implementation.

About the Implementation Interface

The processor implementation class derives from a Studio-generated interface class. There is a single abstract method that the implementation class must implement. The abstract method has the following interface:

```java
public <Processor_Name>Response invoke(
    DiscoveryResolutionProcessorContext context,
    <Processor_Name>Request request) throws ProcessorException
{
}
```
Table 12–1 describes the methods provided to the developer by the first parameter, `context`, outlined in "About the Implementation Interface".

<table>
<thead>
<tr>
<th>Context method</th>
<th>Return Object Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getActionName()</code></td>
<td>String</td>
<td>Getter for the name of this action.</td>
</tr>
<tr>
<td><code>getProcessorName()</code></td>
<td>String</td>
<td>Getter for the name of this processor.</td>
</tr>
<tr>
<td><code>getUnhandledDiscrepancies()</code></td>
<td>Collection Discrepancy</td>
<td>Getter for a list of unprocessed discrepancies for this invocation.</td>
</tr>
<tr>
<td><code>getAllDiscrepancies()</code></td>
<td>Collection Discrepancy</td>
<td>Getter for a list of processed and unprocessed discrepancies for this invocation.</td>
</tr>
<tr>
<td><code>discrepancyProcessed(Discrepancy disc)</code></td>
<td>void</td>
<td>Sets the status of the input discrepancy to OPERATION_PROCESSED.</td>
</tr>
<tr>
<td><code>discrepancyFailed(Discrepancy disc, String failureMessage)</code></td>
<td>void</td>
<td>Sets the status of the input discrepancy to OPERATION_FAILED, and also sets the failure message.</td>
</tr>
<tr>
<td><code>discrepancyReceived(Discrepancy disc)</code></td>
<td>void</td>
<td>Sets the status of the input discrepancy to OPERATION_RECEIVED.</td>
</tr>
</tbody>
</table>

The second parameter, `request`, contains getters for each item in the Input Parameters table. It also contains a getter to retrieve the groups and items listed in the Properties tabbed page.

**Return Type of Invoke()**

The return type of the `invoke()` method varies, depending on the output parameters setting in the Context Parameters tabbed page.

If there is no output parameter, then the return type is void.

If there are one or more output parameters, then the return type is a generated class with the name `Processor_NameResponse`. This Response class has getters and setters for each item in the Output Parameters table.

**About the General Flow of the Discrepancy Resolution Processor**

The usual pattern for implementing a discrepancy resolution processor is as follows:

1. Fetch the list of unhandled discrepancies using `context.getUnhandledDiscrepancies()`.

2. Allocate discrepancies based on logical groupings; for example: all discrepancies on a single card and on its children port.
   
   Keep discrepancies that can be handled by this processor, and ignore or remove other discrepancies.

3. For each group, perform operations to fix the discrepancies. Then, based on operation results, set their status to `Processed` or `Failed`.
   
   An error message can be saved in the Failure Reason field of the discrepancy, which is displayed in the Network Integrity UI.

4. Set output parameters.
Fetching Discrepancies

The discrepancy resolution processor can use the context input parameter to fetch the list of discrepancies to process. In the general flow, the processor uses the method `getUnhandledDiscrepancies()` on `context` to retrieve a list of discrepancies that are not yet handled by any previous processors.

It is also possible to retrieve the original full list of discrepancies by using the method `getAllDiscrepancies()`, but this list includes discrepancies that are already handled by a prior resolution processor.

It is possible to make updates to already handled discrepancies, such as updating the `Notes` field to add more text.

See "Details Page" for a list of fields that can be modified and warnings about which fields should not be modified.

See "About Discrepancies" for more information about the attributes of a Discrepancy object.

Grouping Discrepancies

Usually, a single resolution processor is responsible for handling the discrepancies of a single entity type; for example: logical device or device interface only, or more frequently an explicit set of specifications of an entity type.

Sometimes, a processor specializes in handling discrepancies of a very specific nature. Therefore, the next logical task is to examine each unhandled discrepancy, to determine how it should be handled by this processor.

A processor frequently uses one or more of the following discrepancy attributes as criteria for handling. Of course, it may use all other attributes as criteria for determining special handling, if necessary.

See "About Discrepancies" for a detailed explanation of these attributes:

- **Type**: indicates the error being reported; for example: attribute mismatch, missing entity, and so on.
- **externalEntityType, staticEntityType**: indicates the type and specification of the target entity.
- **attributeOrRelationshipName**: indicates the attribute or the association that has the discrepancy
- **compareValue, referenceValue**: each attribute indicates the value of an attribute on one side of the comparison
- **compareEntity, referenceEntity**: each attribute is a reference to one entity being compared; see "About the Compare and Reference Sides" and "About Discrepancy Types" for important information on what entity each attribute is actually referencing.
- **childTargetEntity**: this is an additional entity reference used only for Association or Entity discrepancy types; see "About Discrepancy Types" for further information.

Handling Discrepancies

Now that the target has been identified and grouped, the processor can decide whether to proceed with the handling. If the processor can resolve this discrepancy,
then the processor can make appropriate API calls necessary to make the desired resolution on the system, and report the result.

See "Reporting the Resolution Result".

Alternatively, the processor can decide to skip the discrepancy, and begin processing the next one. The skipped discrepancy subsequently appears in the unhandled list of discrepancies for the next processor.

**Reporting the Resolution Result**

When a discrepancy has resolved successfully, simply pass this discrepancy into the context using the method `discrepancyProcessed`. This sets the discrepancy status to *Processed*.

```java
context.discrepancyProcessed( discrepancy );
```

If the processor fails to resolve a discrepancy, it should set the discrepancy status to *Failed* using the method `discrepancyFailed` in the context.

This method takes an additional String argument, which the processor can set a short message to be displayed in the UI. The string is stored in the `reasonForFailure` attribute of the discrepancy.

```java
context.discrepancyFailed( discrepancy, "Sample error message.");
```

If the processor needs to make a series of asynchronous invocations to handle a discrepancy, it can set the discrepancy status to *Received* at the end of the first invocation.

This indicates to Network Integrity and to Network Integrity users that the discrepancy resolution is in progress. This is done using the method `discrepancyReceived` in the context.

```java
context.discrepancyReceived( discrepancy );
```

See "About Discrepancy Status" for an explanation of the transition rules for status values.

**Handling Discrepancies Asynchronously**

There are situations in which a discrepancy resolution operation cannot be completed within a single invocation. For example, the CORBA interface for an external system to create a trouble ticket requires the caller to supply a callback object for the notification of the final operation result and ticket ID.

In this example, the resolution processor code can prepare the callback object and make the initial CORBA call to submit the trouble ticket, and then it must return from the `invoke()` method. The subsequent resolution handling code must reside in the callback object, and receives the notification, updating the discrepancy status accordingly.

In such cases, the processor should set the status of the discrepancy to *RECEIVED* using `context.discrepancyReceived()` at the end of the handling code in the processor's `invoke()` method. This indicates to Network Integrity and to Network Integrity users that resolution processing is in progress, and that additional status updates arrive later.

Note: This error message is limited to a maximum of 255 characters.
You must also save the entityID of the discrepancy (using `discrepancy.getEntityId()`) during the processor's `invoke()` method. When the subsequent resolution handling operation reaches its conclusion, the status of the original discrepancy must be updated to `PROCESSED` or `FAILED`. This is done using the Network Integrity Web Services interface by first retrieving the discrepancy using the entityID, and then updating the status of the discrepancy.

The topic of how to save the entityID and how to create the subsequent code invocation is beyond the scope of this guide. You may use any techniques available in J2EE to perform these tasks.
Discovery action scans are often used to scan multiple devices in the network. A discovery action scan can use a variety of protocols to perform a scan. To facilitate scan processing, Network Integrity supports an address expansion and validation software component called an address handler. Address handlers perform two functions:

- They validate that a user supplied address string is syntactically correct for a protocol.
- They expand address strings which represent multiple addresses, into a collection of individual addresses.

This allows the user to configure a scan of multiple addresses using a compact, efficient notation; for example: the notation 10.156.67.1-254 is used to express the range of addresses from 10.156.67.1 to 10.156.67.254, which is 254 addresses.

Discovery actions can optionally specify an address handler to use. It is best practice to create an address handler whenever address validation is desired. Addresses are validated when a scan configuration for the discovery action is saved, and also when the scan is run.

In addition, address strings representing multiple addresses are expanded into a collection of addresses when the scan runs. When an address string is expanded into multiple addresses, Network Integrity calls into the Discovery Action multiple times until each individual address has been scanned. The scanning of multiple addresses is done in parallel.

Address handlers are defined in Design Studio for Network Integrity. Design Studio for Network Integrity generates some artifacts for the address handlers. However, the user needs to supply implementation code to complete the address handler.

Address handlers become stateless session beans in the run-time environment. Cartridges containing an address handler can easily be deployed using Studio or the Oracle Cartridge Deployer tool.

---

**Note:** Address handlers cannot be created in the same cartridge as actions.

---

Network Integrity delivers an IP Address handler and a URL Address handler with the following cartridges:

- Network Integrity MIB-II SNMP Cartridge
  See *Network Integrity MIB-II SNMP Cartridge Guide* for further information.
- Network Integrity - Cisco Router and Switch SNMP Cartridge.
Creating Address Handlers

This section describes how to create an address handler.

To create an address handler:

1. Select New from the Studio menu, then select Integrity, then select Address Handler.

   The Address Handler Wizard appears.

2. In the Project field, select the name of the project that includes the new address handler. The currently selected project appears by default.

3. The Type field lists the data type; Address Handler.

4. In the Name field, enter a name for the new address handler. This is the name that appears in Network Integrity.

   When you define an address handler name, you must ensure that the name does not:
   - Contain special characters
   - Contain javascript reserved keywords
   - Start with a numeral

   For information about defining address handler names, see the following Web page:

5. In the Folder field, select a location for the address handler. If you leave the Folder field blank, the specification is created in the default location.

6. The Implementation Prefix field is a read-only field which is set automatically to a value which is derived from the address handler name.

7. Click Finish.

   Studio creates the new address handler and displays it in the Address Handler editor.

Specifying the Address Handler Implementation

You must specify the implementation class for an address handler.

You do this from the Address Handler editor by doing one of the following:

- From the Details page, click Select to choose an existing class for the element implementation.
  
  In the Select Java Implementation page, enter the type name and matching items, and click OK.

- Create the implementation class by selecting the Implementation Class link.

About the Address Handler Interface

Address Handlers must implement the AddressHandler interface which is shown and described in the following section:
package oracle.communications.integrity.api;
import java.util.List;
import oracle.communications.integrity.common.AddressHandlerException;
import oracle.communications.integrity.common.AddressesStatus;

/**
 * AddressHandler is common interface which should be implemented by the
 * class implementing the Address expansion and validation of addresses.
 */
public interface AddressHandler {

    /**
     * This method expands the list of address or addressRange provided.
     * @param addressRangeList - a list of String representing either an address
     * or an address range
     * @return List - a list of Strings each of which represents an individual
     * address
     * @throws AddressHandlerException
     */
    public List<String> expandAddressRange(List<String> addressRangeList) throws AddressHandlerException;

    /**
     * This method validates the list of address provided.
     * @param address
     * @return AddressesStatus
     * @throws AddressHandlerException
     */
    public AddressesStatus validate(List<String> address) throws AddressHandlerException;

    /**
     * This method validates the single address provided.
     * @param address
     * @return boolean
     * @throws AddressHandlerException
     */
    public boolean validate(String address) throws AddressHandlerException;

    /**
     * This method is used to count the number of addresses after expansion of
     * address parameter passed.
     * Here maxCountLimit can be NULL. If maxCountLimit is NULL, method return the
     * total count of expanded address.
     * If maxCountLimit is specified, method does not count the expanded address
     * beyond that limit and returns the maxCountLimit + 1.
     * @param addressRangeList
     * @param maxCountLimit
     * @return int
     * @throws AddressHandlerException
     */
    public int countExpandedAddresses(List<String> addressRangeList, Integer maxCountLimit) throws AddressHandlerException;
}
About the Address Handlers Cartridge

Network Integrity provides a cartridge called Address Handlers which implements the following address handlers:

- **IPAddressHandler** validates and expands both IPv4 and IPv6 address.
  It validates and expands the following IP address formats:
  - Single IP addresses; for example: 10.156.67.123
  - IP address ranges using “-”; for example: 10.156.67.10-125
  - IP address ranges using “*”; for example: 10.156.67.*, equal to 10.156.67.0-255
  - IP addresses using Classless Inter-Domain Routing (CIDR); for example: 10.156.67.0/24

- **URLAddressHandler** validates URL syntax addresses.

- **File TransferAddressHandler** validates addresses and paths used by the File Transfer processor, as follows.
  - allows the field to contain one or two tokens delimited by "/"
  - using a single token identifies:
    - the absolute path to files that are local to the Network Integrity server, for example: /tmp
  - using two tokens identifies:
    - the remote location and absolute path
    - host_name/path, for example: someserver.us.com/tmp/test
    - IPV4Address/path, for example: 10.156.58.63/tmp/test
    - IPV6Address/path
  - validates the proper format of IPV4 and IPV6Address

---

**Note:** The File Transfer processor does not support address expansion and relative paths.

---

- **Corba URLAddressHandler** validates that the address entered in Network Integrity is a properly formatted IPv4 or IPv6 CorbaLoc URL. For more information, see Network Integrity CORBA Cartridge Guide.
This chapter outlines how to use result categories in Oracle Communications Network Integrity.

### About Result Categories

*Result Category* is a mandatory field for the following action types:

- Discovery action
- Import action
- Assimilation action

*Result Category* is used to identify a result group when an action adds the results to the result group. The *Result Category* value configured for the action must match the result group name in the Java implementation (the `addToResult` method) into which the discovered data is persisted. See Network Integrity Information Model Reference for information about using result categories in modeling results.

For more information about this Java implementation, see "Working with Discovery Actions and Processors".

Design Studio does not explicitly validate this result category name specified in Studio against the actual result group name specified in the Java implementation.

The result category and action define a result source for the following action types:

- Discrepancy detection action
- Discrepancy resolution action

Both actions work on results (to perform discrepancy detection or resolution, respectively) based on the result source.

For example, a discovery action persists discovered data in two result categories:

- *Device*
- *Workstation*

A discrepancy detection action defines the result source as this discovery action and the *Device* result category, that is, this action works on discovered data generated by the discovery action and stored in the *Device* result group. If the result category configured for the discovery action does not match the actual result group name in the Java implementation, but the discovery detection action is configured with the result source based on the result category configured in Studio, the discrepancy detection action is not able to find the results to perform discrepancy detection at run time. In
other words, the result group name does not match the result category defined in result source.

To add results to the result group in the Java implementation, see "Configuring a Discovery Processor".

About Result Categories in Network Integrity

*Result Category* is the identifier for a result group. An action configured with a result category persists the results to the corresponding result group after being deployed and executed in Network Integrity. The result category is visible in the Network Integrity user interface (UI) when displaying the scan results.

*Figure 14–1* shows the result category in the Network Integrity UI. The discovered device is stored under the result category, *Device*.

**Tip:** Provide an appropriate result category when configuring an action, because this value is displayed in the Network Integrity user interface.

*Figure 14–1  Result Category in Network Integrity UI*
Model Extension Using Specifications

This chapter provides information about using specifications to create model extensions in Oracle Communications Network Integrity.

Overview

Network Integrity cartridges persist their results to POMS in an Oracle Communications Information Model representation. The Information Model defines a base set of entities and the relationships between them. The Information Model is extendable through Specifications. Specifications add attributes to the base Information Model entities. It is expected that most cartridges must extend the Information Model entities and therefore must make use of Specifications.

Like UI Parameters, Specifications are defined using the Design Studio data dictionary. UI Parameters are actually a special type of specification. A Specification used for model extension is associated with a single Information Model entity type. Multiple Specification types can be defined for each Information Model entity type. Within the Data Dictionary, a Specification is a special type of data dictionary structure. The elements that comprise the structure are also known as characteristics.

Specifications can be shared between cartridges. Network Integrity ensures that when multiple cartridges are deployed together, their shared Specifications are compatible.

When cartridge code persist information to POMS it creates Information Model entities and usually a specific type of Specification is attached to each Information Model entity to hold additional attributes. Within the Network Integrity UI, an Information Model entity and its specification are represented as a single object.

All action types must define which specification types they use. This is done through the Action’s Model tab. The Model tab defines the list of Model Collections that the Action uses. The Model Collections define the Specifications and implicitly through the Specifications, the base Information Model entity types that the Actions use. Design Studio for Network Integrity generates special classes, called specification helpers, to help simplify the usage of the Specifications by the cartridge developer.

As Specifications attributes are displayed in the Network Integrity UI, the user can define UI Hints for the Specifications. These UI Hints work much like the UI Hints for UI Parameters except that Specification attributes are always read-only in the Network Integrity UI. For Specifications, the following UI attributes can be associated with the following data dictionary item types:

- Structures can be decorated with the following attributes:
  - **Label**: this attribute specifies the label that displays in the UI
Working with Specifications

Working with Specification involves four areas of Design Studio for Network Integrity:

- The data dictionary defines Specifications. The data dictionary defines:
  - The data dictionary structures that are used as Specifications
  - The elements of the data dictionary structure
  - The enumeration types of the elements within the Specification’s structure
- The Model Collections editor chooses the Specifications that are part of a Model Collection.
- The Action editor Model tab is used for choosing the Model Collections that the Action uses. This indirectly defines which Specifications is used.
- The cartridge editor’s UI Hints tab is where UI attributes are defined for the data dictionary structures and child elements of the Specification.

The high level steps for using Specifications in Actions are:

1. Use the data dictionary to define a Specification.
2. Refer to the Specifications from a Model Collection.
3. From the Action’s Model tab choose the Model Collections that the Action uses.
4. Navigate to the cartridge UI Hints tab to add UI attributes to the Specification data dictionary structures and elements.

Data Dictionary Editor

Use of the data dictionary editor is described in Design Studio documentation, Working with the Data Dictionary section. A Specification is a special type of data dictionary structure. To define a Specification:

1. Create a top level structure for the Specification.
2. On the Details tab specify the Display Name.
3. On the Entity Type tab specify:
   - Dynamic
   - Choose the base entity type of the Specification. The chooser contains an entry for each base entity type that can be extended. It also contains an entry UI Parameter Specification which is a special entry for UI Parameters. This has nothing to do with extending the Information Model; it should not be chosen.
4. Add child elements to the specification. For the child element specify:
• Display Name.
• Type: The Child elements must be of type string or an enumeration type of base type string.

5. Save the data dictionary.

As Specifications can be shared between Actions, it is not always necessary to define a new Specification.

Model Collections

To make use of Specifications in your cartridges specify which Specifications the Cartridge and Actions use. Do this with Model Collections and the Action Model tab.

Create Model Collection

To create a Model Collection:
1. In the Cartridge View, from the popup menu, select New then Integrity then Model Collection. A Model Collection Wizard dialog displays.
2. In the Collection Wizard dialog specify:
   • Project: the project in which the Model Collection is created.
   • Name: the name of the Model Collection.
   • Folder: to group different Model Collections into different folders, enter the name of a new folder, or choose an existing folder by using the Browse button. The folders can be found inside the Model Collections folder of the project tree in the Cartridge view.
3. Click Finish to create the new Model Collection. The Model Collection opens in a Model Collection editor.

Adding a Specification to a Model Collection

To add a Specification to a Model Collection:
1. Select Add in the Model Collections editor. A Data Item Selection dialog displays.
2. Use the Specification Type, Dictionary, Type and Search fields to filter the Specifications displayed in the dialog.
3. Select the desired Specifications and click the OK. The Data Item Selection dialog closes.
4. Save the Model Collection.

Removing a Specification from a Model Collection

To remove a Specification from a Model Collection:
1. Select the Specification and click Remove.
2. Save the Model Collection.

Action Model Tab

From the Action’s Model tab define which Model Collections the Action should use.
Adding a Model Collection to a Model

To add a Model Collection to the model, use the following procedure:

1. Select Add. A Select the Model Collection to Add dialog displays.

2. In the dialog, you can filter the Model Collections that are displayed by entering the first few characters of the Model Collections name.

3. Select the desired Model Collection and select the OK. The Model Collection are added to the Action’s model.

4. Save the Action.

Removing a Model Collection from an Model

To remove a Model Collection from the model, use the following procedure:

1. Select the Model Collection, and click Remove.

2. Save the Action.

Cartridge Editor UI Hints Tab

UI Hints are defined for Specifications from the Cartridge editor’s UI Hints tab. The user assigns UI attributes to the data dictionary structures and child elements using the fields at the right of the tab.

For Specifications, UI Hints apply to all Network Integrity UI screens where the Specifications are displayed. For Specifications, only the Label and Tool Tip UI attributes can be associated with the data dictionary items.

The procedures for defining UI Hints for Specifications are the same as for UI Parameters. See "Working with UI Parameter Examples" for further details.

Specification Usage Examples

This section describes some common operations involving Specifications.

Create and Use Specification Example

The steps to create and use a Specification are:

1. Create the new Specification using the procedure described in "Data Dictionary Editor".

2. Refer to the new Specification from a Model Collection.
   a. If you must create a Model Collection, then use the procedure described in "Create Model Collection".
   b. Use the procedure described in "Adding a Specification to a Model Collection" to refer to the Specification.

3. If necessary, refer to the Model Collection from the Action’s Model tab. Use the procedure described in "Adding a Model Collection to a Model".

Use an Existing Specification Example

The steps to use an existing Specification are:

1. Refer to the Specification from a Model Collection. See "Adding a Specification to a Model Collection".
2. If necessary, refer to the Model Collection from the Action’s Model tab. Use the procedure described in "Adding a Model Collection to a Model".
3. Change UI hints for the Specification from the Cartridge UI Hints tab. See "Cartridge Editor UI Hints Tab" for details.

Modify Specification Usage Example

The steps to modify a Specification are:

1. Modify the Specification’s data dictionary structure. Use of the data dictionary editor is described in the Design Studio documentation, Working with the Data Dictionary section.
2. If desired, modify the UI Hints for the Specification. See "UI Hints Tab" for details.

Defining Enumerated Values for Specifications Example

To define Specifications which have enumerated values, the following steps are required:

1. In the data dictionary, create a top level element of type string.
2. In the Enumerations tab for the top level element, define the set of enumerations. Set both the code and the description. The code becomes an enumerated value for the Specification. The description becomes the label displayed in the Network Integrity UI.
3. On the element which supports the enumeration, change the type to be the type of the top level element just defined.

Stop Using a Specification Usage Example

To stop using a Specification, use the following steps:

1. Remove the Specification from the Model Collection. See "Removing a Specification from a Model Collection" for the procedure.
2. If the Model Collection contains no other specifications you may to delete it. First remove it from any model’s that are referring to it. Then delete it.
3. If the Specification is not being used elsewhere in the cartridge, you should remove any UI hints for the specification from the Cartridge UI Hints tab. Use the following procedure:
   a. From the Cartridge UI Hints tab, select the Specification and the select the Remove all UI Hints for the Selected Elements.
   b. Save the Cartridge.
   c. Select Clean UI Hints.
   d. If the Specification is not being used, it is a best practice to remove it from the Data Dictionary.
Specification Helpers

Specification Helper classes are generated by the Design Studio for Network Integrity when Specifications are referred to by Model Collections. The Specification Helpers are generated to the following package:

- **Cartridge Default Package.Model Collection Name.Model Collection Name**

The names of the Specification Helpers are based on the names of the Specifications in the Model Collection. For example if the name of the Specification is `deviceGeneric`, then the name of the Specification Helper is `DeviceGeneric`.

The Specification Helpers have a getter and setter method for each element in the specification. The Specification Helper also has a constructor which takes a POMS entity interface object. A code sample which illustrates the use of a Specification Helper is shown below. In the code sample, the DeviceGeneric class is the Specification Helper.

```java
// create a Logical Device entity which uses 
// the Device Generic specification.
//
LogicalDevice logicalDevice = PersistenceHelper.makeEntity(LogicalDevice.class);
DeviceGeneric logicalDeviceExt = new DeviceGeneric(logicalDevice);

// Set static attribute values to LogicalDevice.
logicalDevice.setId(makeLDevID(scanResponse));
logicalDevice.setName(rfc1213Mib.getSysName());
logicalDevice.setDescription(rfc1213Mib.getSysDescr());

// Set dynamic attributes/characteristics.
//
logicalDeviceExt.setMgmtIPAddress(scanResponse.getManagementIP());
logicalDeviceExt.setSysObjectId(rfc1213Mib.getSysObjectID());
```
This chapter provides information about how the persistent object modeling service (POMS) manages persistent data in Oracle Communications Network Integrity.

About POMS

POMS manages all persisted data within Network Integrity. Developers use POMS for most cartridge development, but they rarely need to deal explicitly with persistence details.

POMS includes the Java definition of the entities and relationships described in Oracle Communications Information Model Reference and Network Integrity Information Model Reference.

While POMS includes both interface and implementation classes for the entities, developers work only with interfaces. These interfaces provide getters and setters for attributes and relationships. Use the PersistenceHelper POMS SDK class to instantiate entities.

Developers can use the POMS SDK Finder class to find and retrieve existing persisted entities.

POMS is built on the EclipseLink Java persistence API (JPA) platform. Developers do not usually need to know EclipseLink or JPA to use the POMS SDK. The exception is find operations where developers may have to know Java Persistence Query Language (JPQL). See "Working with the POMS Finder" for more information about the find operations.

Table 16–1 describes the POMS SDK APIs.

<table>
<thead>
<tr>
<th>Table 16–1 POMS SDK API Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>POMS SDK APIs</strong></td>
</tr>
<tr>
<td>Entities</td>
</tr>
<tr>
<td>Specifications and characteristics</td>
</tr>
<tr>
<td>PersistenceHelper</td>
</tr>
<tr>
<td>Finder</td>
</tr>
</tbody>
</table>
Working with POMS Entities

The POMS Java interface for an entity has the same name as the entity described in the model document. For example, entity Equipment becomes:

```java
public interface Equipment
```

Attributes are accessed with familiar Java getters and setters. For example, the Equipment name attribute is defined by:

```java
public java.lang.String getName();
public void setName( java.lang.String name );
```

An entity may contain enumerated values for certain attributes. POMS implements these as Java enumerations. For example, the EMSServiceState from LogicalDevice has the following:

```java
public enum EMSServiceState {
    UNKNOWN( "UNKNOWN" ),
    IN_SERVICE( "IN_SERVICE" ),
    OUT_OF_SERVICE( "OUT_OF_SERVICE" ),
    TESTING( "TESTING" ),
    IN_MAINTENANCE( "IN_MAINTENANCE" );

    public oracle.communications.inventory.api.entity.EMSServiceState
        getNativeEmsServiceState();
    public void setNativeEmsServiceState( oracle.communications.inventory.api.entity.EMSServiceState
        nativeEmsServiceState );
```

When creating results, for example in a Discovery Processor, developers must instantiate POMS entities. Use the PersistenceHelper class, passing the desired entity class to the makeEntity method:

```java
Equipment equipment = PersistenceHelper.makeEntity(Equipment.class);
```

Working with POMS Relationships

Related entities are also accessed with getters and setters.

One-to-one Relationships

When a relationship refers to a single entity, the entity is accessed directly. For example, the mapped logical device on a physical device:

```java
public oracle.communications.inventory.api.entity.LogicalDevice
    getMappedLogicalDevice();
public void setMappedLogicalDevice( oracle.communications.inventory.api.entity.LogicalDevice
    mappedLogicalDevice );
```

One-to-Many or Many-to-Many Relationships

When a relationship can refer to multiple entities, the entities are accessed through a collection. For example, the equipment to physical port relationship:

```java
public java.util.List<oracle.communications.inventory.api.entity.PhysicalPort>
    getPhysicalPorts();
public void setPhysicalPorts( java.util.List<oracle.communications.inventory.api.entity.PhysicalPort>
    physicalPorts );
```
A getter never returns `null` for the collection. If there are no related entities, an empty collection is returned. That means the developer can safely add entities without creating a collection. For example:

```java
equipment.getPhysicalPorts().add(physicalPort);
```

## Ordered and Unordered Relationships

POMS uses a `List` for the collection because the Oracle Communications Information Model defines an ordered relationship for physical ports on equipment. In other cases, order does not matter and so POMS uses a `Set` for the collection. For example, the parent relationship from `Equipment` to `EquipmentHolder`:

```java
public java.util.Set<oracle.communications.inventory.api.entity.EquipmentHolderEquipmentRel> getParentEquipmentHolders();
public void setParentEquipmentHolders(java.util.Set<oracle.communications.inventory.api.entity.EquipmentHolderEquipmentRel> equipmentHolders);
```

## Bi-directional Relationships

Certain relationships in the model are bi-directional. POMS includes accessors on entities on both sides of a bi-directional relationship, and the relationship can be set from either side. The physical device to logical device relationship described in the "One-to-one Relationships" example is bi-directional. The other side of this relationship, on the logical device, is defined as:

```java
public java.util.List<oracle.communications.inventory.api.entity.PhysicalDevice> getMappedPhysicalDevices();
public void setMappedPhysicalDevices( java.util.List<oracle.communications.inventory.api.entity.PhysicalDevice> mappedPhysicalDevices );
```

This is a many-to-one relationship, so there is a collection on the logical device side and single entity on the physical device side. To relate a physical and logical device, developers can either set from the physical device:

```java
physicalDevice.setMappedLogicalDevice(logicalDevice);
```

or set from the logical device:

```java
logicalDevice.getMappedPhysicalDevices ().add(physicalDevice);
```

## Relationship Entities

In some cases, the model defines an intermediate relationship entity instead of relating two entities directly. For example, the Information Model defines `EquipmentEquipmentRel` to relate two pieces of equipment. To create this type of relationship, instantiate the relationship entity and set the related entities. For the equipment to equipment example:

```java
EquipmentEquipmentRel parentEquipmentRel = PersistenceHelper.makeEntity(EquipmentEquipmentRel.class);
parentEquipmentRel.setChildEquipment(equipment);
parentEquipmentRel.setParentEquipment(parentEquipment);
```

## Working with Specifications and Characteristics

Developers can use the generated specification helper classes to avoid directly dealing with specifications and characteristics. See "Model Extension Using Specifications" for
Developers can determine if an entity supports characteristics and specification by referencing the model documentation, or by checking the POMS interface. Entities that support characteristics and specifications extend the CharacteristicExtensible interface. For example:

```java
oracle.communications.inventory.api.CharacteristicExtensible <oracle.com munications.inventory.api.entity.EquipmentCharacteristic>;
```

The specification and characteristics are related entities like any other, characteristics being multi-valued:

```java
public oracle.communications.inventory.api.entity.EquipmentSpecification
getSpecification();
public void setSpecification( oracle.communications.inventory.api.entity.
EquipmentSpecification specification );

public java.util.Set<oracle.communications.inventory.api.entity.
EquipmentCharacteristic> getCharacteristics();
public void setCharacteristics( java.util.Set<oracle.communications.inventory.api.
entity.EquipmentCharacteristic> characteristics );
```

As a convenience, POMS also lets developers access a characteristic by name through the map returned by `getCharacteristicMap`:

```java
public java.util.Map<String, oracle.communications.inventory.api.entity.
EquipmentCharacteristic> getCharacteristicMap();
```

---

**Working with the POMS Finder**

Developers can use the POMS Finder to retrieve previously persisted data, however, developers do not typically need to use the Finder.

The most basic use of the Finders is "Find by Entity". More powerful and flexible queries are possible with the Java Persistence Query Language (JPQL). Developers can also control whether entities are returned completely or a with a subset of attributes. Developers can also use paging to return data in manageable chunks where queries might return a large volume of data.

**Find by Entity**

To find entities matching an example entity, instantiate an entity of the appropriate type and set one or more attributes. Use the `findByExample` method to return a collection of matching entities. Here is an example that looks for the specification for a Cisco 3640 physical device:

```java
Finder finder = PersistenceHelper.makeFinder();
PhysicalDeviceSpecification example =
    PersistenceHelper.makeEntity(PhysicalDeviceSpecification.class);
    example.setName("Cisco3640");

Collection<PhysicalDeviceSpecification> specifications =
    finder.findByEntity(example, 'name');
if (specifications.size() == 1) {
    System.out.println("found specification");
}
```
Find by JPQL

Java Persistence Query Language (JPQL) is a powerful way to express queries. The following examples can be understood without knowing JPQL, especially if the developer is familiar with SQL; however, developers must learn JPQL to build their own queries.

For an introduction to JPQL, use the following link:


To perform a JPQL query use the following workflow:

1. Instantiate a Finder.
2. Initialize any parameters (these parameters are bound to variables in the JPQL expression).
3. Specify the desired result type.
4. Use the `findByJPQL` method to return matching results.

In following example queries, the first is equivalent to the example in the section "Find by Entity", and returns a particular specification. The second uses a join in the JPQL expression to return all physical devices that use this specification.

```java
Finder finder = PersistenceHelper.makeFinder();
finder.addParameter("name", "Cisco3640");
finder.setResultClass(PhysicalDeviceSpecification.class);
Collection<PhysicalDeviceSpecification> specifications = finder.findByJPQL(
    "SELECT o FROM PhysicalDeviceSpecification o " +
    "WHERE o.name = :name");

finder.setResultClass(PhysicalDevice.class);
Collection<PhysicalDevice> cisco3640Devices = finder.findByJPQL(
    "SELECT o FROM PhysicalDevice o JOIN o.specification s " +
    "WHERE s.name = :name");
```

A JPQL query does not need to return complete entities. It can return one or more attributes from matched entities. To return only name and ID from a physical device, the developer would modify the previous example as follows:

```java
Collection cisco3640Devices = finder.findByJPQL(
    "SELECT o.name,o.id FROM PhysicalDevice o JOIN o.specification s " +
    "WHERE s.name = :name");
for (Object device : cisco3640Devices) {
    Object[] attributes = (cisco3640DevicesObject[]) device;
    System.out.println("Found Cisco 3640 named " + attributes[0] + " with id " +
        attributes[1]);
}
```

The code snippet also shows how to iterate over the results. Since the returned type is not a POMS entity, the attribute values are available as `Object` arrays. Developers would not set the result class in this case.

While JPQL and the Finder support operations that modify persisted data (update, delete, and so on), developers should never modify POMS data with JPQL. The Finder is intended only for read operations.

Find with Paged Results

When working with a large number of entities, process them in smaller batches to reduce memory usage. The Finder supports paged results. Initialize the Finder
normally, then specify the range of value to retrieve. This modifies the original physical device example to page through devices 20 at a time:

```java
int pageSize = 20;
int start = 0;
while (true) {
    finder.setRange(start, start + pageSize - 1);
    Collection<PhysicalDevice> cisco3640Devices = finder.findByJPQL(
        "SELECT o FROM PhysicalDevice o JOIN o.specification s WHERE s.name = :name";
    for (PhysicalDevice device : cisco3640Devices) {
        System.out.println(device.getName());
        if (cisco3640Devices.size() < pageSize) {
            break;
        }
        start += pageSize;
    }
}
```

### POMS SDK Interfaces

The following are the **PersistenceHelper** API methods:

```java
public static < E extends Object > E makeEntity( Class< E > entity);
public static oracle.communications.platform.persistence.Finder makeFinder( ) ;
```

The following are the **Finder** API methods:

```java
/** *
 * Set the result Class to query.
 *
 * @param resultClass
 *            the interface of each result in the result set
 */
public void setResultClass(Class resultClass);

/** *
 * Set the range of the result set to return, starting of the zero-based
 * start index and ending at the end index, exclusive. For example,
 * setRange(0,5) returns 5 results indexed at 0 thru 4.
 *
 * @param start
 *            zero-based start index
 * @param end
 *            ending index, exclusive
 * @see javax.jdo.Query#setRange
 */
public void setRange(long start, long end);

/** *
 * Add the parameter name and value that are used to define the filter.
 *
 * @param names
 *            Parameter names beginning with an underscore ('_') are illegal. They may
 *            conflict with additional parameters used internally by this Finder.
 */
```

16-6   Developer's Guide
### About Persist Results

The persistResults method is available in the context of discovery, import and assimilation scan action types. This method persists in-memory result entities to the database and invalidates the entities. You may or may not need to explicitly call this method, depending on the sort of results that your action produces for a given invocation.

If the result set is small (for example, one result group for a particular device), then there is no need to call this method. Your result entities are automatically persisted when the action completes.

If the result set is large (for example multiple devices imported from an inventory system), call persistResults to write the information to the database, reducing memory...
consumption. In the context of an import action, you would likely want to call the persistResults after results for each device are modeled.

Since persistResults invalidates any in-memory entities, you should not hold a reference to any result entity across a call to persist results.
Working with UI Parameters

Design Studio supports user interface (UI) parameters on its discovery, import, and assimilation action types. UI parameters become input fields in the Network Integrity user interface into which you enter values when defining scans. These values are passed to the scan’s processors when the scans run. UI parameters allow scans for a specific action to be configured for different environments and situations.

Network Integrity uses Studio data dictionaries to implement UI parameters. See Studio documentation for a description of data dictionaries.

Data dictionaries contain data dictionary items. Data dictionary items are of two types, elements and structures. A UI parameter is defined by a data dictionary structure. A data dictionary structure is a complex type which can have multiple child elements and multiple child structures. Child elements can be of a variety of different types: string, int, boolean, and so on. When used for UI parameters however, the data dictionary structure is restricted to child elements, that is, it cannot contain child structure. In addition, elements are restricted to type string or an enumeration of base type string.

Associating Data Dictionary Item Types with Attributes

You use Studio to decorate data dictionary items with GUI display and control information so that the UI parameters can be rendered and given appropriate handling in Network Integrity.

**Note:** UI attributes for top level elements are not used.

The following section lists UI attributes that can be associated with data dictionary item types:

- **Associating UI Attributes with Data Dictionary Structures**
- **Associating UI Attributes with Data Dictionary Child Elements**

**Associating UI Attributes with Data Dictionary Structures**

Structures can be decorated with the following attributes:

- **Label:** this attribute specifies the label to be displayed in the UI
- **Tool Tip:** this attribute specifies a short message to be displayed when the pointer hovers over the field
Associating UI Attributes with Data Dictionary Child Elements

Child elements can be decorated with the following attributes:

- **Label**: this attribute specifies the label to be displayed in the UI.
- **Tool Tip**: this attribute specifies a short message to be displayed when the pointer hovers over the field.
- **Control Type**: this attribute specifies the control type to be used for the UI input field. Choose from:
  - *Input Text*. If this value is selected, the field accepts user entered text
  - *Choice*. If this value is selected, the user can select from a set of values
- **Mandatory**: this attribute specifies whether it is mandatory to enter a value into the UI input field.
- **Default Value**: this attribute specifies the default value for a UI input field.
- **Secret**: this attribute specifies whether the UI field is given special handling such that its value is not displayed to the user. Secret is specified as true to provide special fields such as fields containing passwords.
- **Read Only**: this attribute specifies whether a UI field is read-only.

Displaying UI Parameters in the Network Integrity User Interface

When UI Parameter data dictionary items are created, the Display Name field is copied into the UI Hints label. Subsequent changes to the Display Name field are not propagated to the UI Hints label.

UI Parameters are displayed on two different pages in the Network Integrity UI:

- The Create Scan page. See Figure 17–1.
- The Scan Details page. See Figure 17–2.

Studio defines the separate UI attribute settings for each of the pages. 

*Figure 17–1* shows the Create Scans page.
Figure 17–1  The Create Scans Page

Figure 17–2 shows the Scan Details page.

Figure 17–2  The Scan Details Page
Working with UI Parameters

Working with UI parameters involves three separate Studio areas:

- About the UI Parameters Tab
- About the Data Dictionary
- About the UI Hints Tab

You must perform several high-level steps to use UI Parameters in actions. See "Using UI Parameters in Actions".

About the UI Parameters Tab

For action types which support UI parameters, their editors contain a UI Parameters pane. The action types which support UI Parameters are:

- The Discovery action
- The Import action
- The Assimilation action

The UI Parameters tab is used for choosing the data dictionary structures that become the UI parameters of the action.

The UI Parameters tab is also used to navigate to the other areas of Studio related to UI parameters.

Figure 17–3 shows the UI Parameters tab for a Discovery action.

Figure 17–3  The UI Parameters Label

About the Data Dictionary

The data dictionary is used to define the following data dictionary items used by the UI parameters:

- The UI parameter structure
- The elements of the UI parameter structure
- The enumerations types of the elements within the UI parameter structure
Figure 17–4 shows the data dictionary editor.

Figure 17–4  The Data Dictionary Editor

About the UI Hints Tab

The cartridge editor’s UI Hints tab is where UI attributes are defined for the data dictionary structures and child elements of the UI Parameters. Figure 17–5 shows the UI Hints tab.

Figure 17–5  The UI Hints Tab
Using UI Parameters in Actions

To use UI Parameters in actions:

1. In the action editor, select the **UI Parameters** tab.
2. Select an existing data dictionary structure or create a data dictionary structure.

---

**Note:** Although data dictionary structures for UI Parameters can be created directly in the data dictionary editor, this is not recommended for Network Integrity. Data dictionary structures for UI Parameters should be created using the UI Parameters tab.

---

3. Navigate to the data dictionary to define the child elements of the structure.
4. Use the data dictionary to define top level elements for enumeration types.
5. Go to the **UI Hints** tab for the cartridge to add UI attributes to the UI Parameter data dictionary structures and elements.

About the Action Editor UI Parameters Tab

The Action editor UI Parameters tab contains the following buttons:

- **New.** Click to create a data dictionary structure for use as a UI parameter. A Create Data Dictionary Item dialog appears. This is a standard Design Studio dialog. To use this dialog, see Design Studio documentation.

- **Select.** Click to select an existing data dictionary UI Parameters structure. A Data Item Selection dialog appears. This is a standard Design Studio dialog. To use this dialog, see Design Studio documentation.

- **Remove.** Select the data dictionary structures to be removed from the list used by the Action, and then click **Remove.** The data dictionary structures are not removed from the data dictionary.

- **Move Up.** Select the data dictionary structure to move up the list. Click **Move Up.** The list order does not affect the presentation order in the Network Integrity UI.

- **Move Down.** Select the data dictionary structure to move down the list. Click **Move Down.** The list order does not affect the presentation order in the Network Integrity UI.

The UI Parameters list also has a popup menu which supports the following functions:

- **Select UI Parameter from Data Dictionary.** This menu item provides the same functionality as the **Select** button:

  Click to select an existing data dictionary UI Parameters structure. A Data Item Selection dialog appears. This is a standard Design Studio dialog. To use this dialog, see Design Studio documentation.

- **Create new UI Parameter in Data Dictionary.** This menu item provides the same functionality as the **New** button:

  Click to create a data dictionary structure for use as a UI parameter. A Create Data Dictionary Item dialog appears. This is a standard Design Studio dialog. To use this dialog, see Design Studio documentation.

- **Open UI Parameter in Data Dictionary.** This menu item opens the selected UI Parameter in its data dictionary.
Using UI Parameters in Actions

- **Open UI Parameter in UI Hints.** This menu item opens the selected UI Parameter in the cartridge UI Hints tab.
- **Remove.** Select the data dictionary structures to be removed from the list used by the Action, and then select **Remove**. The data dictionary structures are not removed from the data dictionary.
- **Expand.** This menu item expands the structure of the selected UI Parameters.
- **Collapse.** This menu item collapses the structure of the selected UI Parameters.

**About the Data Dictionary Editor**

Use of the data dictionary editor is described in Design Studio documentation. When used for UI Parameters, the data dictionary structure can contain only child elements. It cannot contain child structures. Elements are restricted to type string or an enumeration of base type string.

**About the Cartridge Editor UI Hints Tab**

By selecting a UI Parameter in the action editor UI Parameters tab, and then selecting the **Open UI Parameter in UI Hints** menu item, you can navigate to this tab with the selected UI Parameter in focus.

Assign UI attributes to the data dictionary structures and child elements using the fields at the right of the tab.

When assigning UI attributes to data dictionary elements, you must first select the appropriate Network Integrity page where the UI attributes are used. This is done using the **Page/Panel** list at the top right of the tab. The valid choices are:

- For data dictionary structures, choose **Default**. These UI attributes apply to all Network Integrity pages. The only UI attributes used are Label and Tool Tip.
- For data dictionary elements, choose **Create Scan/Plug-in Panel** or **Scan Detail/Plug-in Panel**. You must define the UI attributes for both Network Integrity pages.
  
  There is a third option, which is **Default**. This should never be used for Network Integrity UI Parameters.

The valid choice for structures and elements of other entity types is **Default/Default**. The only UI Structures used are Label and Tool Tip.

**Working with UI Parameter Examples**

This section describes how to perform some common operations using UI Parameters:

- Creating and Using UI Parameters
- Using Existing UI Parameters
- Editing UI Parameters
- Defining a UI Parameters Enumerated Values Example
- Deleting a UI Parameter

**Creating and Using UI Parameters**

To create and use a UI Parameter:

1. Create a data dictionary structure:
Using UI Parameters in Actions

a. From the UI Parameter tab for the action, click **New**.
   A Create Data Dictionary Item dialog appears.

b. In the **Name** field, enter the name of the UI Parameter. Default values are valid for all of the other fields.

c. Click **Save**. A data dictionary structure is created.

d. Save the action.

2. Define the fields for the UI parameter:

a. From the UI Parameter tab for the action, double-click the new data dictionary structure.
   The data dictionary editor opens with the focus on the relevant structure dialog.

b. If an enumerated set of values is required for a child element in the data dictionary structure, define a top level element, of type **string**, and define the enumerated values using the **Enumerations** tab.
   This element defines a new type to define structure child elements in the next step.

c. Define the child elements for the UI Parameters structure. Enter values for the **Name**, the **Display Name** and the **Type** fields. Defaults values are valid for the remaining fields.
   Only string type or enumerations of base type string for UI Parameter elements are valid. For child elements that should be restricted to an enumerated set of values, specify the type to be the same as the type defined in the previous step.

d. Save the data dictionary.

3. Define the UI hints for the UI parameter:

a. From the UI Parameter tab for the action, navigate to the Cartridge editor **UI Hints** tab by right-clicking the UI Parameter, and then selecting **Open UI Parameter in UI Hints**.

b. Define the **Label** and **Tool Tip** UI attributes for the UI Parameter structure.

c. Define UI attributes for each element in the UI Parameter. Do this for both the **Create Scan** and **Scan Detail** pages.
   For each element, perform the following:
   Select the Create Scan page, and set all of the UI attributes.
   Select the Scan Detail page, and set all of the UI attributes.

d. Save the cartridge.

**Using Existing UI Parameters**
To use an existing UI Parameter:

1. Select an existing data dictionary structure:

a. From the UI Parameter tab for the action, click **Select**.
   A Data Item Selection dialog appears. See Design Studio documentation for information on configuring this dialog.

b. Select the required UI Parameter. Click **OK**.
c. Click OK. The UI Parameter is selected.
d. Save the action.

2. Define the UI hints for the UI parameter:
   a. From the UI Parameter tab for the action, navigate to the Cartridge editor **UI Hints** tab by right-clicking the UI Parameter, and then selecting **Open UI Parameter in UI Hints**.
   b. Define the Label and Tool Tip UI attributes for the UI Parameter structure.
   c. Define UI attributes for each element in the UI Parameter. Do this for both the **Create Scan** and **Scan Detail** pages.
      For each element, do the following:
      Select the Create Scan page, and set all of the UI attributes.
      Select the Scan Detail page, and set all of the UI attributes.
   d. Save the cartridge.

**Editing UI Parameters**

To modify an existing UI Parameter:

1. Modify the UI Parameter’s data dictionary structure:
   a. Double-click the UI parameter. The data dictionary editor appears.
   b. Edit the details of the data dictionary structure.
   c. Save the changes.

2. Modify the UI hints for the UI parameter:
   a. From the UI Parameter tab for the action, the UI Parameter, and then selecting **Open UI Parameter in UI Hints**.
   b. (Optional) Edit the Label and Tool Tip UI attributes for the UI Parameter structure.
   c. (Optional) Modify the UI attributes for each element in the UI Parameter. Do this for both the **Create Scan** and **Scan Detail** pages.
      For each element, do the following:
      Select the Create Scan page, and edit UI attributes.
      Select the Scan Detail page, and edit UI attributes.
   d. Save the cartridge.

**Defining a UI Parameters Enumerated Values Example**

To define UI Parameters with enumerated values:

1. In the data dictionary, create a top level element of type **string**:
2. Select the **Enumerations** tab.
3. Set the code and the description. The code becomes an enumerated value UI Parameter. The description becomes the label displayed in the Network Integrity UI.
4. On the UI Parameter element, change the type to the type of the top level element you have defined.
Deleting a UI Parameter

To delete a UI Parameter:

1. Remove the UI parameters from the action:
   a. From the UI Parameter tab for the action, select the UI parameters to be deleted.
   b. Click Remove.
   c. Save the action.

2. If no other actions in the cartridge use the UI Parameter deleted in step 1, remove the UI hints for the UI parameter:
   a. From the UI Parameter tab for the action, navigate to the Cartridge editor UI Hints tab by right-clicking the UI Parameter, and then selecting Remove all UI Hints for the Selected Elements.
   b. Save the cartridge.
   c. Click Clean UI Hints.
   d. If no other cartridges use the UI parameter, and if you do not expect to use it again in the future, remove the UI Parameter from the data dictionary.
Oracle Design Studio is used to set conditions for processors used in action executions in Oracle Communications Network Integrity.

About Conditions

An action can contain conditions. By creating and applying conditions to processors, at run time you can dynamically control which processors should be executed inside an action based on the condition (whether true or false).

Conditions are implemented as a Java class that implements the Condition Interface. When configuring a condition for an action, Design Studio code-generates the Condition Interface. The Studio designer then implements this Condition Interface.

When a condition is configured and implemented, the condition can be applied to one or more processors. For each processor, the same condition can be set to be either true or false. One processor can also have multiple conditions applied. In this case, the processor are executed if all the conditions are true.

To create conditions, see Network Integrity Studio Help.

To find out more information about the class generation and implementation, see:
- About Generated Classes and the Implementation Class

To apply conditions to processors, see the following:
- Applying Conditions to Processors

To import conditions, see the following:
- Importing Actions with Conditions

About Generated Classes and the Implementation Class

When a condition is configured for an action, Design Studio code-generates two classes:
- Condition Interface, which takes the name ConditionName_Implementation_Prefix\Condition.java
- Request, which takes the name ConditionName_Implementation_Prefix\Request.java

The generated classes are available at:
Studio_Workspace\NI_Project_Root\generated\src\Project_Default_Package\Action_Type\Action_Implementation_Prefix
The following is a sample generated condition interface which defines one method, `checkCondition`. In this sample, `ValidDeviceRequest` is the generated request class for the condition:

```java
public interface ValidDeviceCondition {

    /**
     * @param context
     * @param request
     * @return @see boolean
     * @throws ProcessorException
     */
    public boolean checkCondition(DiscoveryProcessorContext context,
                                ValidDeviceRequest request) throws ProcessorException;
}
```

Design Studio also generates the skeleton implementation class for this condition interface. To open the Java editor and start the Java implementation, click the Implementation Class link.

### Applying Conditions to Processors

You apply conditions to processors to control the execution flow of the processors inside the action.

To apply a condition to a processor, see Network Integrity Studio Help.

Multiple conditions can be applied to a processor, which means this processor is executed only if all the conditions are met. One condition can be applied to one or more processors, which means that same condition can be applied to multiple processors.

### Importing Actions with Conditions

If an imported action contains conditions, these conditions are also replicated after importing. The imported conditions cannot be removed from the applied processor, which is also imported along with the action.

See "Adding an Existing Action" for information on importing an existing action to another action.

Figure 18–1 shows an example of an imported action, showing processors and associated conditions.

---

**Note:** This directory also contains generated Action MDB and Controller classes.
In Figure 18–1, Processor2 is imported along with its two conditions:

- ValidDevice
- DemoCondition

These conditions are available to local processors, but cannot be removed. In this example, ValidDevice is selected, but the Remove button is disabled.

You can add a new condition to the imported processor. In this example, a new condition, DummyCondition, is added to Processor2. Any new condition added to the imported processor can be removed.

By adding new conditions to the imported processors, you can change the behavior of an imported action by using conditions to control whether an imported processor is executed.

Creating Condition Examples

This section outlines examples of setting conditions in Network Integrity:

- Multi-Vendor Discovery
- Multi-Protocol Discovery

**Multi-Vendor Discovery**

See "Multiple Vendor SNMP Discovery".

**Multi-Protocol Discovery**

See "Multiple Protocol Discoveries".
This chapter provides information about the extensibility SDK for Oracle Communications Network Integrity.

About Extensibility Scenarios

Cartridges and Actions in Network Integrity are extensible using Design Studio for Network Integrity. The productized and sample cartridges provided by Network Integrity are designed to be completely extensible and re-usable. The following sections are step-by-step examples of some common extensibility scenarios.

Each of the scenarios follows a detailed example but is meant to demonstrate the many extensibility features and methods within Network Integrity cartridge development. The following concepts are being demonstrated in the scenarios:

- Re-using existing Actions
- Conditional Execution using Conditions
- The use of Specifications and Characteristics to extend the model
- The use of input and output parameters
- The use of UI parameters
- Using filters to modify default discrepancy detection behavior
- What extension points are available in productized cartridges

The extensibility scenarios are:

- **Extending MIB II SNMP Discovery for Updated Vendor and Interface Type**
  Describes the steps required to update the vendor number and interface type mapping tables in the MIB II SNMP Discovery cartridge

- **Extending the Existing Cartridge to Discover and Reconcile New Characteristics**
  Describes the steps required to extend an existing cartridge to discover new data from a device and reconcile this data with an Inventory system.

- **Extending the MIB II SNMP Discovery to Change Interface Name Value**
  Describes the steps required to extend the MIB II SNMP Discovery Action to map the SNMP variable ifName to the interface entity name rather than the entity interface description.

- **Multiple Vendor SNMP Discovery**
  Describes the steps required to extend an existing cartridge to discover data from devices from multiple vendors.
Multiple Protocol Discoveries

Describes the steps required to extend an existing cartridge to discover data using multiple protocols.

Extending MIB II SNMP Discovery for Updated Vendor and Interface Type

This scenario describes the steps required to update the vendor number and interface type mapping tables in the MIB II SNMP Discovery cartridge. The vendor number table translates an enterprise object identifier number to a vendor name. The interface type table translates an ifType value into a human readable name. These mapping tables are created by the MIB II Properties Initializer, and are made available to the following processors by its output parameters. Although the properties were up to date at the time of cartridge creation, the industry updates them from time to time with additions or changes.

The following tasks are performed in this example:

- adds a new interface type (#333, “tachyonEther”),
- adds a new vendor number (#90210, “West Beverly Hills School District”), and
- changes an existing vendor name (#34416, from “Ottawa Area Intermediate School District” to “Ottawa Area Middle School District”).

The following high-level steps are involved:

- Creating a new Network Integrity Cartridge Project
- Creating a new Discovery Action that re-uses the MIB II SNMP discovery action
- Creating a new Discovery Processor inside the new Action to update the mapping tables.

Prerequisites:

- The following cartridges are loaded into the Design Studio workspace and are building without errors
  - Address_Handlers
  - MIB_IIModel
  - MIB_IISNMP_Cartridge
- The above cartridge projects can be loaded by importing the MIB II SNMP Cartridge ZIP file into Design Studio. For instructions on importing cartridge zip files, refer to "Exporting and Importing Cartridges".

Creating a New Project, Discovery Action, and Discovery Processor

Figure 19–1 shows the Discovery Action created in this scenario:
To update and discover the MIB II SNMP Discovery cartridge, use the following procedure:

1. Use the following steps:
   a. Create a cartridge to hold the new discovery action.
   b. Create a Network Integrity Cartridge Project called **Vendor_Type_Update_Cartridge**.
   c. See "Creating a Cartridge" to for instructions on how to create a Network Integrity Cartridge Project.

2. Create a Discovery Action called **Discover Updated MIB II SNMP**. Refer "Working with Discovery Actions and Processors" for instructions on creating a new Discovery Action, and extending existing Discovery Actions.

3. On the Processor tab of the new Discovery Action select **Add** and add the **Discover MIB II SNMP** action. This adds all the processors from the **Discover MIB II SNMP** action.

4. Now a new Discovery Processor is required to make updates to the two mapping tables. Use the following steps:
   a. Select **New** to add a new Discovery Processor called, for example, **MIB II Properties Updater**.
   b. Select **Move Up** to move the new processor up to row 2, below **MIB II Properties Initializer** and above **MIB II SNMP Collector**.
   c. Refer to "Creating Discovery Processors" for instructions on how to create a Discovery Processor, and how to move a processor.

5. Open the editor for the **MIB II Properties Initializer Updater** Discovery Processor and select the **snmpIfTypeMap** and **snmpVendorNameMap** output parameter from the **MIB II Properties Initializer** as input parameters. Refer to "About Context Parameters" for instructions on how to add input parameters to a processor.

6. Create the implementation class for this discovery processor. Refer to "About Processor Implementation" for instructions on how to add an implementation class to a processor.

7. Add the implementation code into the body of the invoke() method of the discovery processor implementation class, similar to the following:
   ```
   // Rename 34416 from 'Ottawa Area Intermediate School District'
   // to 'Ottawa Area Middle School District'
   // Add a new vendor ID 90210 = West Beverly Hills School District
   ```
Map<String, String> vendorNameMap = request.getSnmpVendorNameMap();
vendorNameMap.put("34416", "Ottawa Area Middle School District");
vendorNameMap.put("90210", "West Beverly Hills School District");

// Add a new interface type 333 as tachyonEther.
//
Map<String, String> ifTypeMap = request.getSnmpIfTypeMap();
ifTypeMap.put("333", "tachyonEther");

Extending the Existing Cartridge to Discover and Reconcile New Characteristics

This scenario describes the steps required to extend an existing cartridge to discover new data from a device and reconcile this data with an Inventory system.

The following high-level steps are involved:

- Create a Network Integrity Cartridge Project
- Create Specifications and Characteristics for the new data
- Create a Discovery Action that re-uses existing discovery action
- Configure the SNMP discovery of this data
- Map this data to the Oracle Communications Information Model
- Create a Discrepancy Detection and Discrepancy Resolution Actions that re-use existing actions
- (optional) Create a Import Action that re-uses existing import action

For this scenario the following data is discovered and stored on the Physical Device:

- Running Configuration Last Saved Date
- Running Configuration Last Modified Date
- Startup Configuration Last Modified Date

Prerequisites:

- The following cartridge projects are loaded into the Design Studio workspace and are building without errors:
  - Base_Detection_Cartridge
  - Address Handlers
  - Cisco_Model
  - Cisco_SNMP_Cartridge
  - Cisco_UIM_Cartridge
  - Cisco_UIM_Model
  - MIB_II_Model
  - MIB_II_SNMP_Cartridge
  - MIB_II_UIM_Cartridge

- The above cartridge projects can be loaded by importing the Cisco SNMP Cartridge, MIB II UIM Cartridge, and Cisco UIM Cartridge ZIP files into Design Studio. For instructions on importing cartridge zip files refer to “Exporting and Importing Cartridges”.

19-4  Developer's Guide
Creating Project, Discovery Action with Processors, and Adding Characteristics and Specifications

To perform an extended discovery action, use the following procedure:

Figure 19–2 shows the Extended Discovery Action created in this scenario:

1. In the Design Studio Cartridge View create a Network Integrity Cartridge Project, for example named Extensibility_Cartridge. Refer to "Creating a Cartridge" for instructions on creating a Network Integrity Cartridge Project.

2. Copy the existing specifications and add the new characteristics. Use the following steps:
   a. Double-click cisco3640Router. This is the data dictionary in the Cisco_UIM_Model.
   b. Right-click cisco3640 structure and select Copy To -> Extensibility_Cartridge.
   c. Double-click cisco6509Router data dictionary.
   d. Right-click cat6509 structure and select Copy To -> Extensibility_Cartridge.
   e. Double-click cisco7206VXRRouter data dictionary.
   f. Right-click cisco7206VXR structure and select Copy To -> Extensibility_Cartridge.

3. Rename the copied specifications and have new characteristics added to them. Double-click Extensibility_Cartridge to open the data dictionary.

4. Because specification names have to be unique in the system, the name of the cisco3640 specification needs to be changed to support new characteristics. To rename the specification use the following steps:
   a. Right-click the cisco3640 structure.
   b. Select Rename (or press F2).
   c. Give the specification a new name, for example add the word Custom to the end of the name.
   d. Refer to "Model Extension Using Specifications" for more details about specifications and characteristics.

5. Add the new characteristics to the new specification to hold the data that is discovered from the device. To add the new characteristics:
   a. Right-click the cisco3640Custom structure.
   b. Select Add Child Element.
c. Specify a name for the characteristic, for example `runningConfigLastSavedDate`.
d. Right-click the `cisco3640Custom` structure.
e. Select Add Child Element.
f. Specify a name for the characteristic, for example `runningConfigLastChangedDate`.
g. Right-click the `cisco3640Custom` structure.
h. Select Add Child Element.
i. Specify a name for the characteristic, for example `startupConfigLastChangedDate`.

6. Repeat steps 4 and 5 for structures `cisco7206VXR` and `cat6509` in the Extensibility_Cartridge data dictionary.

7. A model collection entity is required to indicate which specifications this cartridge uses. Create a Model Collection called Extensibility Model Collection in the Extensibility_Cartridge project. Refer to "Model Extension Using Specifications" for instructions on how to create a Model Collection and add specifications to it.

8. Add the Physical Device specifications `cisco3640Custom`, `cisco7206VXRCustom` and `cat6509Custom` (created and modified in the steps above) to the new Model Collection.

9. Save all changes.

10. Now a new Discovery Action is required in the Cartridge Project that discovers the device and models the new fields. To do this create a Discovery Action, for example called Discover Extended Cisco. Refer to "Creating a Discovery Action" for instructions on how to create a Discovery Action and how to extend existing Discovery Actions.

11. On the Processor tab of the new Discovery Action select Add and add the Discover Enhanced Cisco SNMP action. This adds all the processors from the Discover Enhanced Cisco SNMP action.

12. A new SNMP Processor is required to retrieve the new fields from the device. To create a SNMP processor select New and add a new SNMP Processor, for example called Custom Cisco Collector. Refer to "Creating an SNMP Processor" for instructions on how to create a SNMP Processor.

13. Perform a web search for the CISCO-CONFIG-MAN-MIB, download a copy of the MIB and manually copy the CISCO-CONFIG-MAN-MIB to the MIB directory (specified in Windows then Preferences then Oracle Design Studio then Network Integrity).

14. The new Custom Cisco Collector needs the CISCO-CONFIG-MAN-MIB file. Load the CISCO-CONFIG-MAN-MIB into the SNMP Processor MIB browser. Refer to "Supporting New MIBs" on how to add a new MIB file to Design Studio.

15. The new MIB also needs to be added to the server side. Copy CISCO-CONFIG-MAN-MIB file to the SNMP Adapter on the Network Integrity server side. Refer to "Extending the SNMP JCA Resource Adapter" section for details.

16. Add the following MIB Objects from CISCO-CONFIG-MAN-MIB.private.

```plaintext
enterprises.cisco.ciscoMgmt.ciscoConfigManMIB.ciscoConfigManMIBObjects.ccmHistory:
```
Extending the Existing Cartridge to Discover and Reconcile New Characteristics

- ccmHistoryRunningLastChanged
- ccmHistoryRunningLastSaved
- ccmHistoryStartupLastChanged

17. Save all changes.

18. Now a new Discovery Processor is required that maps the newly discovered fields to the specifications and characteristics defined earlier in the scenario. To do this click New and add a new Discovery Processor, for example called Custom Cisco Modeler. Refer to "Creating Discovery Processors" for instructions on how to add a new Discovery Processor.

19. Select the input parameter physicalDevice that is output by the Cisco SNMP Physical Modeler. Refer to "About Context Parameters" for instructions on how to add input parameters to a processor.

20. Select the output document of the Custom Cisco Collector as an input parameter.

21. Create the implementation class for this discovery processor. Refer to "About Processor Implementation" for instructions on how to add an implementation class to a processor.

22. Add the following implementation code into the invoke() method that was auto-generated:

```java
// Get the running config and startup config values from the SNMP response document
// Keep the values in local variables
CiscoConfigManMibMib configMib = request.getCustomCiscoCollectorResponseDocument().getDiscoveryResult().getCiscoConfigManMibResults();
String runningConfigChanged = Long.toString(configMib.getCcmHistoryRunningLastChanged());
String runningConfigSaved = Long.toString(configMib.getCcmHistoryRunningLastSaved());
String startupConfigChanged = Long.toString(configMib.getCcmHistoryStartupLastChanged());
if (request.getPhysicalDevice() != null) {
    // Get the physical device.
    PhysicalDevice physicalDevice = request.getPhysicalDevice();

    // Get the specification name on the physical device
    String specName = physicalDevice.getSpecification().getName();
    if (specName != null) {
        // Change the specification to the custom specification type
        // and set the new fields
        if (specName.equals(Cisco3640.SPEC_NAME)) {
            Cisco3640Custom custom = new Cisco3640Custom(physicalDevice);
            custom.setRunningConfigLastChangedDate(runningConfigChanged);
            custom.setRunningConfigLastSavedDate(runningConfigSaved);
            custom.setStartupConfigLastChangedDate(startupConfigChanged);
        } else if (specName.equals(Cat6509.SPEC_NAME)) {
```
Cat6509Custom custom = new Cat6509Custom(physicalDevice);
custom.setRunningConfigLastChangedDate(runningConfigChanged);
custom.setRunningConfigLastSavedDate(runningConfigSaved);
custom.setStartupConfigLastChangedDate(startupConfigChanged);
} else if (specName.equals(Cisco7206VXR.SPEC_NAME)) {
    Cisco7206VXRCustom custom = new Cisco7206VXRCustom(physicalDevice);
custom.setRunningConfigLastChangedDate(runningConfigChanged);
custom.setRunningConfigLastSavedDate(runningConfigSaved);
custom.setStartupConfigLastChangedDate(startupConfigChanged);
}
}

Creating a Discrepancy Detection Action with Processors

Figure 19–3 shows the Discrepancy Detection Action created in this scenario:

1. Create a Discrepancy Detection Action called Detect Extended Cisco Discrepancies. Refer to “Working with Discrepancy Detection Actions and Processors” for instructions on how to create a Discrepancy Detection Action and how to extend existing Discrepancy Detection Actions.

2. On the Result Source tab, click Add and add the Discover Extended Cisco action. This indicates that the extended Discrepancy Detection action applies to extended discovery results.

3. On the Processor tab of the new Discrepancy Detection Action select Add and add the Detect Enhanced Cisco Discrepancies action. This adds all the processors from the Detect Enhanced Cisco Discrepancies action.

Creating a Discrepancy Resolution Action with Processors

Figure 19–4 shows the Discrepancy Resolution Action created in this scenario:
Extending the Existing Cartridge to Discover and Reconcile New Characteristics

Figure 19–4 Discrepancy Resolution Action

1. Create a Discrepancy Resolution Action called Resolve Extended Cisco in UIM. Refer to "Working with Discrepancy Resolution Actions and Processors" for instructions on how to create a Discrepancy Resolution Action, and how to extend existing Discrepancy Resolution Actions.

2. On the Details panel, select Correct in UIM as the Resolution Action Label. This indicates that discrepancies are corrected in the UIM inventory system.

3. On the Result Source tab, click Add and add the Discover Extended Cisco action. This indicates that the extended Discrepancy Resolution action applies to extended discovery results.

4. On the Processor tab of the new Discrepancy Resolution Action select Add and add the Resolve Cisco in UIM action. This adds all the processors from the Resolve Cisco in UIM action.

5. Create UIM Physical Device specifications that match the cisco3640Custom, cisco7206VXRCustom and cat6509Custom specifications already created for Network Integrity. Refer to the UIM document, Understanding Unified Inventory Management, Release 7.1, for details on how to use Design Studio to create and deploy UIM specifications.

6. Each UIM specification should contain the Information Model characteristics and the three new custom characteristics (runningConfigLastSavedDate, runningConfigLastChangedDate, and startupConfigLastChangedDate).
   a. Each UIM specification should set the same parent child and cardinality relationship as the original UIM specifications cisco3640, cisco7206VXR, and cat6509. (The first child needs to be set to the chassis, and inherits the remaining cascade relationships).
   b. Select the existing specification, copy, paste with rename, and add the new characteristics.

Creating and Import Action with Processors (Optional)

Figure 19–5 shows the option Import Action created in this scenario:
This is an optional section. The existing Import Cisco from UIM action available in the Cisco UIM Cartridge is used to import the extended devices types with new characteristics. Create this import action if you want to deploy the Extensibility Cartridge without also deploying the Cisco UIM Cartridge.

1. Create a Import Action called Import Extended Cisco from UIM. Refer to "Working with Import Actions and Processors" for instructions on how to create a Import Action, and how to extend existing Import Actions.

2. On the Processor tab of the new Import Action select Add and add the Import Cisco from UIM action. This adds all the processors from the Import Cisco from UIM action.

Extending the MIB II SNMP Discovery to Change Interface Name Value

This scenario describes the steps required to extend the MIB II SNMP Discovery Action to map the ifName to the interface name rather than the interface description. In addition, this scenario exposes a UI parameter that the end-user can use to control the behavior of the interface name mapping.

Note: Changing how the name field is mapped has repercussions on how generic discrepancy detection looks up import entities because the lookup is done using name field (this can be modified using discrepancy detection filters, refer to "About Filters" for details). If the interface name field is modified for discovery, but is not modified on the import data many ‘extra entity’ discrepancies are produced because discrepancy detection is unable to find the interface of the import side.

This problem can be avoided by ensuring that the name field for discovery and import are identical, or by using a different field other than name to look up the interface on the import side. An example of using a different field is in the Detect MIB II UIM Discrepancies Discrepancy Detection Action in the MIB_II_UIM_Cartridge. This action overrides the default lookup to use the NativeEMSName instead of the name field.

Step 16 in this scenario describes how the Detect MIB II UIM Discrepancies Discrepancy Detection Action can be re-used in this extensibility scenario.
Extending the MIB II SNMP Discovery to Change Interface Name Value

- Create new Network Integrity Cartridge Project
- Create new Discovery Action that re-uses existing discovery action
- Configure new UI Parameter
- Add new processor to change mapping of interface name

Prerequisites:
- The following cartridges are loaded into the Design Studio workspace and are building without errors:
  - Address_Handlers
  - MIB_II_Model
  - MIB_II_SNMP_Cartridge
- The above cartridge projects can be loaded by importing the MIB II SNMP Cartridge ZIP file into Design Studio. For instructions on importing cartridge ZIP files, refer to "Exporting and Importing Cartridges".

Creating a Discovery Action with Processor

**Figure 19–6** shows the Discovery Action created in this scenario:

**Figure 19–6  Discovery Action**

1. Create a cartridge to hold the new discovery action. Create a Network Integrity Cartridge Project called **InterfaceName_Cartridge**. Refer to "Creating a Cartridge" for instructions on how to create a Network Integrity Cartridge Project.
2. Create a Discovery Action called **Discover Custom MIB II SNMP**. Refer to "Configuring a Discovery Action" for instructions on how to create a Discovery Action and how to extend existing Discovery Actions.
3. On the Processor tab of the new Discovery Action click **Add** and select the **Discover MIB II SNMP** action. This adds all the processors from the **Discover MIB II SNMP** action.
4. On the UI Parameters tab click **New** and add a new UI Parameter structure called **MIBIIParameters**. Refer to "Working with UI Parameters" for instructions on creating and configuring UI Parameters.
5. Double-click the **MIBIIParameters** UI parameter structure to open the data dictionary.
6. In the data dictionary create a top-level element called **mapIfDescToInterfaceName**. Use the following steps:
a. Right-click the data dictionary.

b. Select Add Element.

c. Enter mapIfDescToInterfaceName for the element name.

d. Click OK.

7. Add two enumeration values to this element by clicking the Enumerations tab and then clicking the Add button. Add the enumerations true and false.

8. Then add a child element to the MIBIICustomParameters structure called mapIfDescToInterfaceName and select the type mapIfDescToInterfaceName created in step 6.

---

**Note:** Enumerations in specifications must be added as described in steps 7 and 8. If the enumeration values are added directly on the Child Element, the enumeration values are not generated correctly in the generated UI Parameters page.

---

9. Open the Cartridge Editor for the InterfaceName_Cartridge and click the UI Hints Tab.

10. Find the MIBIICustomParameters structure and click the child mapIfDescToInterfaceName element.

11. Select Create Scan page on the right pane.

12. Change the Label to 'Map Description to Interface Name'.

13. Change Control Type to Choice.

14. Change Default Value to true.

15. Save all changes.

16. Open the editor for Discover Custom MIB II SNMP discovery action and select the Processors tab.

17. Select New and add a new Discovery Processor called Custom Interface Name Modeler. Refer to "Creating Discovery Processors" for instructions on how to create a Discovery Processor.

18. Open the editor for the Custom Interface Name Modeler Discovery Processor. Choose the Context Parameters tab and select the output parameter logicalDevice from the MIB II SNMP Modeler as an input parameter. Refer to "About Context Parameters" for instructions on how to add input parameters to a processor.

19. Select the Details tab and create the implementation class for the discovery processor.

20. Add the implementation code similar to the following:

```java
@override
public void invoke(DiscoveryProcessorContext context,
                    CustomInterfaceNameModelerProcessorRequest request)
    throws ProcessorException {
```
// if the user specified they do not want the ifDesc as the name of the interface
// then use the ifName instead
if (!"false".equalsIgnoreCase(request.getMibiiCustomParameters().getMapIfDescToInterfaceName())) {
    List<DeviceInterface> deviceInterfaces = request.getLogicalDevice().getDeviceInterfaces();
    changeInterfaceNameToIFName(deviceInterfaces);
}

private void changeInterfaceNameToIFName(List<DeviceInterface> deviceInterfaces) {
    // loop through every interface and change the mapping.
    for (DeviceInterface deviceInterface : deviceInterfaces) {
        // the Discover MIB II SNMP Discovery Action is inserting the ifName into the VendorInterfaceNumber so the following code copies that to the name field
        deviceInterface.setName(deviceInterface.getVendorInterfaceNumber());
        // Change interface name on any sub-interfaces as well
        changeInterfaceNameToIFName(deviceInterface.getSubInterfaces());
    }
}

21. To register Discrepancy Detection and Discrepancy Resolution on the new Discover Custom MIB II SNMP Discovery Action, add new Result Sources to the Detect MIB II UIM Discrepancies and Resolve MIB II in UIM in the MIB_II_UIM_Cartridge that register for results from the Discover Custom MIB II SNMP Discovery Action. Refer to "Working with Discrepancy Detection Actions and Processors" for details. (Alternatively, the Detect MIB II UIM Discrepancies and Resolve MIB II in UIM Actions could be extended in the InterfaceName_Cartridge. Refer to "Extending the Existing Cartridge to Discover and Reconcile New Characteristics" Extensibility Scenario for details on doing this).

Multiple Vendor SNMP Discovery

This scenario describes the steps required to extend an existing cartridge to discover data from devices from multiple vendors.

There are multiple scenarios, depending on the user's objectives.

One scenario is that a user wants to discover devices from a single vendor, for example, Huawei. The user should extend the MIBII SNMP cartridge by reusing the Discover MIB II SNMP Action and adding a Huawei SNMP Collector and a Huawei SNMP Modeler. The Huawei SNMP Collector polls Huawei specific MIBs and the Huawei SNMP Modeler models the Huawei devices based on the collected Huawei SNMP OIDs.

Another scenario is the user wants to discover Juniper devices in addition to Cisco devices. The user should extend the Enhanced Cisco SNMP Action in the Cisco UIM cartridge. This scenario is detailed in this section.

Prerequisites:

- The following cartridges are loaded into the Design Studio workspace and are building without errors:
The above cartridge projects can be loaded by importing the Cisco SNMP Cartridge, MIB II UIM Cartridge, and Cisco UIM Cartridge ZIP files into Design Studio. For instructions on importing cartridge zip files, refer to "Exporting and Importing Cartridges".

Use the sysObjectId from RFC1213MIB to determine a vendor of devices. For example, Cisco devices have the sysObjectld value that starts with 1.3.6.1.4.1.9., and Juniper device have the sysObjectld value starting with 1.3.6.1.4.1.2636. Set up a range of IP addresses and scan those IP addresses by polling the sysObjectld. Based on the sysObjectValue returned, configure two Conditions: one returns true if the sysObjectld value starting with 1.3.6.1.4.1.9. (meaning it is a Cisco device), or return false if otherwise; the other return true if the sysObjectld value starting with 1.3.6.1.4.1.2636. (meaning it is a Juniper device), or return false if otherwise.

The Cisco UIM Cartridge contains the Discover Enhanced Cisco SNMP Action. Create a Discovery Action by reusing this Discover Enhanced Cisco SNMP Action, which gives this new Discovery Action all the functions to discover the enhanced Cisco devices (including the MIB II SNMP discovery). Extend this Discovery Action to support Juniper devices by creating a Juniper SNMP collector and a Juniper modeler to this Discovery Action. The two conditions determine when to execute the Cisco related collectors and modelers and when to execute the Juniper collector and modeler based on the device type.

Create a Discovery Action with Processors and Conditions

Figure 19–7 shows the Discovery Action created in this scenario:

1. Manually copy the JUNIPER-MIB to the MIB directory. The new Juniper SNMP collector needs this JUNIPER-MIB file. Please refer to "Supporting New MIBs" for information on adding a new MIB file to Design Studio.
2. Copy this JUNIPER-MIB file to the SNMP Adapter on Network Integrity server side. Refer to "Extending the SNMP JCA Resource Adapter".

3. Create a Discovery Action, for example **Discover Multi-Vendor**, by reusing the sample **Discover Enhanced Cisco SNMP** Action. Refer to "Adding an Existing Action" for information on reusing an existing Action.

4. Create a Juniper SNMP Collector and add the new SNMP Discovery Processor to the **Discover Multi-Vendor** Discovery Action as the last processor.

5. Add the OID, `jnxBoxDescr` (from JUNIPER-MIB), to this Juniper SNMP Processor. In realistic scenario, more OIDs are polled from this JUNIPER-MIB to model a Juniper device. In this example, only the description field is polled. Refer to "Creating an SNMP Processor" for information on creating and configuring an SNMP Discovery Processor.

6. Create a Juniper modeler and add this new Discovery Processor to the **Discover Multi-Vendor** Discovery Action after the Juniper SNMP collector. This new processor takes the SNMP output parameter from the Juniper SNMP collector as its input parameter. Refer to "Creating Discovery Processors" for information on creating and configuring this Discovery Processor. Implement this Discovery Processor by implementing the `invoke` method. In this example, only the description field for the Juniper device is logged. In a realistic scenario, the complete model the Juniper device would exist in this `invoke` method.

   The following is the Java snippet for the `invoke` method.

   ```java
   @Override
   public void invoke(DiscoveryProcessorContext context, 
                     JuniperProcessorProcessorRequest request) throws ProcessorException {
       logger.log(Level.INFO, "Processing Juniper device " + request.getScopeAddress());
       JuniperSNMPCollectorResponseType responseDoc = request.getJuniperSNMPCollectorResponseDocument();
       DiscoveryResultType result = responseDoc.getDiscoveryResult();
       JuniperMibMib juniperMibResults = result.getJuniperMibResults();
       if(juniperMibResults != null) {
           logger.log(Level.INFO, "Juniper Device Description: " + juniperMibResults.getJnxBoxDescr());
       }
   }
   ```

7. Create a Cisco Condition, that checks the sysObjectId to determine whether a device is a Cisco device or not. This Cisco Condition takes the `mibiisnmpCollectorResponseDocument` (an output parameter from MIB II SNMP Collector) as the input parameter. Refer to "Applying Conditions to Processors" for information on creating a Condition. The following is a Java snippet for this Cisco Condition:

   ```java
   public class CiscoConditionImpl implements CiscoCondition {
       private static final String CISCO_PREFIX = "1.3.6.1.4.1.9.";
       @Override
       public boolean checkCondition(DiscoveryProcessorContext context, 
                                      CiscoRequest request) throws ProcessorException {
           MIBIISNMPCollectorResponseType snmpResponse = request.getMibiisnmpCollectorResponseDocument();
           logger.log(Level.INFO, "GPE CiscoConditionImpl discovery succeeded";
           return (snmpResponse != null && snmpResponse.getDiscoveryStatus() == DiscoveryStatus.SUCCESS);
       }
   }
   ```
if (snmpResponse.getDiscoveryResult().getRfc1213MibResults() != null) {
    String sysObjectId = snmpResponse.getDiscoveryResult().
        .getRfc1213MibResults().getSysObjectID();
    logger.log(Level.INFO, "GPE CiscoConditionImpl raw sys object id: " +
                sysObjectId);
    if (sysObjectId != null) {
        if (sysObjectId.startsWith(".")) {
            sysObjectId = sysObjectId.substring(1);
        }
        return sysObjectId.startsWith(CISCO_PREFIX);
    }
    return false;
}

8. Create a Juniper Condition, that checks the sysObjectId to determine whether a
device is a Juniper device. This Juniper Condition takes the
mibiisnmpCollectorResponseDocument (an output parameter from MIB II
SNMP Collector) as the input parameter. Refer to "Applying Conditions to
Processors" for information on creating a Condition. This Juniper Condition is
similar to the Cisco Condition that was created in step 5. The difference is that the
sysObjectId for Juniper device starts with 1.3.6.1.4.1.2636.

9. Apply the Cisco Condition to the following Cisco Processors and set Equals to
ture:
   a. Cisco SNMP Logical Collector
   b. Cisco SNMP Physical Collector
   c. Cisco SNMP Logical Modeler
   d. Cisco SNMP Physical Modeler
   e. Cisco Enhanced Modeler

   Refer to "Applying Conditions to Processors" for information on applying
   Conditions to Processors. By applying the Cisco Condition to those Processors
   (listed above), those Processors are invoked if the Cisco Condition returns true,
   which means it is a Cisco device.

10. Apply the Juniper Condition to the following two Juniper Processors and set
    Equals to true:
   a. Juniper SNMP Collector
   b. Juniper Processor

**Note:** In this example, only a single Juniper OID is collected and the
value of the collected Juniper OID in the Juniper SNMP Modeler is
logged. In a realistic scenario, several Juniper OIDs are collected to
model a Juniper device. Refer to "Extending the Existing Cartridge to
Discover and Reconcile New Characteristics" on how to map new
SNMP OIDs to new Characteristics and how to update UIM related
Actions for importing, discrepancy detection and resolution with the
new Characteristics.
Multiple Protocol Discoveries

This scenario describes the steps required to extend an existing cartridge to discover data using multiple protocols.

Prerequisites:

- The following cartridges are loaded into the Design Studio workspace and are building without errors:
  - Base_Detection_Cartridge
  - Address_Handlers
  - Cisco_Model
  - Cisco_SNMP_Cartridge
  - Cisco_UIM_Cartridge
  - Cisco_UIM_Model
  - MIB_I_Model
  - MIB_I_SNMP_Cartridge
  - MIB_I_UIM_Cartridge

- The above cartridge projects can be loaded by importing the Cisco SNMP Cartridge, MIB II UIM Cartridge, and Cisco UIM Cartridge ZIP files into Design Studio. For instructions on importing cartridge zip files, refer to "Exporting and Importing Cartridges".

In this scenario, a range of devices can be discovered. Some devices are SNMP-enabled; some devices support an alternate protocol (for example, TL1). With a list of IP addresses for each of these devices, the Discovery Action can dynamically discover a device using either SNMP or the alternate protocol.

The Cisco UIM Sample cartridge contains the sample Discover Enhanced Cisco SNMP Action. Create a Discovery Action by reusing this Discover Enhanced Cisco SNMP Action. This gives the new Discovery Action all the functions to discover an enhanced Cisco device (including the MIB II SNMP discovery). This Discovery Action can be extended to support the alternate protocol by creating a Discovery Processor that implements the alternate protocol to this Discovery Action. To use a JCA resource adapter for this alternate protocol, refer to "JCA Resource Adapters".

Create a Condition, that checks whether the SNMP polling to a device is successful or not. If a device supports SNMP, this Condition returns true; otherwise if the device supports the alternate protocol, this Condition returns false. By applying this Condition to the Processors, the Discovery Action can dynamically discover a device using either SNMP or the alternate protocol.

Create a Discovery Action with Processors and Conditions

Figure 19–8 shows the Discovery Action created in this scenario:
1. Create a Discovery Action, **Discover MultiProtocol**, by reusing the sample **Discover Enhanced Cisco SNMP** Action (refer to "Adding an Existing Action" for information on reusing an existing Action).

2. Create a Discovery Processor, for example **Alternate Protocol Collector**, which implements the alternate protocol to discover a device and add this new Discovery Processor to the **Discover MultiProtocol** Discovery Action as the last Processor. Refer to "Working with Processors", and "Working with Discovery Actions and Processors" for information on creating and configuring this Discovery Processor. Implement this Discovery Processor by implementing the `invoke` method. In this example, one line is logged indicating that this Processor implements an alternate protocol. In a realistic scenario, implement the alternate protocol to discover a device in this `invoke` method. The following is the Java snippet for the `invoke` method.

   ```java
   @Override
   public void invoke(DiscoveryProcessorContext context, AlternateProtocolCollectorProcessorRequest request) throws ProcessorException {
       logger.log(Level.INFO, "SNMP Failed - using alternate protocol to discover device " + request.getScopeAddress());
   }
   ```

3. Create an `SnmpSucceeds` Condition, that checks the SNMP results from MIB II Collector to determine whether the SNMP discovery on a device is successful or not. This `SnmpSucceeds` Condition takes the `mibiisnmpCollectorResponseDocument` (an output parameter from MIB II SNMP Collector) as the input parameter. Refer to "Applying Conditions to Processors" for information on creating a Condition. The following is a Java snippet for this `SnmpSucceeds` Condition:

   ```java
   public class SnmpSucceedsConditionImpl implements SnmpSucceedsCondition {
       @Override
       public boolean checkCondition(DiscoveryProcessorContext context, SnmpSucceedsRequest request) throws ProcessorException {
           MIBIISNMPCollectorResponseType snmpResponse = request.getMibiisnmpCollectorResponseDocument();
           return snmpResponse != null && snmpResponse.getDiscoveryStatus() == DiscoveryStatus.SUCCESS;
       }
   }
   ```

4. Apply the `SnmpSucceeds` Condition to the following Processors and set the `Equals` to be `true`:
   - MIB II SNMP Modeler
   - Cisco SNMP Logical Collector
c. Cisco SNMP Physical Collector

d. Cisco SNMP Logical Modeler

e. Cisco SNMP Physical Modeler

f. Cisco Enhanced Modeler

Refer to "Applying Conditions to Processors" for information on applying Condition to Processors. By applying the SnmpSucceeds Condition to those Processors (listed above), those Processors are invoked if the **SnmpSucceeds** Condition returns **true**, which means it is an SNMP-enabled device.

5. Apply the **SnmpSucceeds** Condition to the following Processor and set the **Equals** to be **false**:

a. Alternate Protocol Collector

By applying the SnmpSucceeds Condition to the Processor (listed above), the Processor is invoked if the **SnmpSucceeds** Condition returns **false**, which means it is not an SNMP-enabled device and therefore supports an alternate protocol.

---

**Note:** In this example, only the message is logged to indicate that an alternate protocol is used in the **Alternate Protocol Collector** Processor. In a realistic scenario, the alternate protocol would be implemented and the network data collected using this protocol and model the collected network data. Refer to "Extending the Existing Cartridge to Discover and Reconcile New Characteristics" section on how to map the collected network data (in that section, the network data is SNMP OID) to new Characteristics and how to update UIM related Actions for importing, discrepancy detection and resolution with the new Characteristics.
Design Studio packages project information into cartridges that can be deployed into Network Integrity.

Cartridges can be developed by customers, systems integrators, Professional Services staff, and third-party vendors.

This chapter describes the cartridge build process, and outlines how to package a cartridge:

- Building Cartridges Automatically
- About the Project Build Order
- About Build Artifacts
- Viewing and Solving Problems in the Problems View
- Reviewing the Error Log

**Building Cartridges Automatically**

It is recommended that the **Build Automatically** setting is enabled in Design Studio. Building automatically means that an incremental build is carried out in Design Studio when a save action is carried out on any Network Integrity entity.

To enable the **Build Automatically** setting:

- From the **Project** menu in Design Studio, check **Build Automatically**.

If **Build Automatically** is not enabled, you have to manually start a build on a Network Integrity cartridge. Design Studio does not automatically perform a build even if there is a change in the project.

To disable **Build Automatically** setting:

- From the **Project** menu, uncheck **Build Automatically**.

**About the Project Build Order**

When Design Studio builds a Network Integrity project, the build process takes place in the following order:

- Generation of Java source code - Generators are invoked to generate Java source codes from Network Integrity models, EJB descriptor files, XML schemas for the SNMP processor, and the Meta Model XML file.

- Java Source Compilation - Eclipse compiles the Java source (including generated Java source and implemented Java source) into classes.
- Building - Builders are invoked to build UI hints, the Data Dictionary, and specifications.

- Validation - Validators are invoked to validate Network Integrity model entities. Validation errors are raised and an error marker displayed on the related entities in Design Studio. If any validation errors are raised, the packaging stage does not take place.

- Packaging - Packagers are invoked to package the cartridge deployment model XML file, the UI hints Metadata Archive (MAR) file, specification Data Access Object (DAO) files, dependent JAR files, the manifest file for JAR files library for EJB, and the final IAR (Integrity Archive) file for the Network Integrity cartridge.

**About Build Artifacts**

Design Studio generates various build artifacts for a Network Integrity cartridge after a successful build. The generated directories are listed in the following order in the directory structure:

- **Out**: This directory contains all the compiled Java classes.

- **Generated**: contains the following build artifacts:
  - Generated Java sources for actions and processors. If the project is sealed without Java source, the JAR file is displayed instead
  - SNMP schema artifacts for the SNMP processor

- **mdsArtifacts**: contains the UI hints artifacts generated by the POMS plug-in.

- **cartridgeBuild**: contains various build artifacts for the Network Integrity cartridge.

- **cartridgeBin**: contains the final packaged Network Integrity cartridge as an IAR file which can be deployed to the Network Integrity server through the Cartridge Management Web Service (CMWS).

The following directories comprise the normal directory structure for a Network Integrity project. Please do not modify these directories:

- **dataDictionary**: contains the Data dictionary

- **doc**: contains documents

- **integrityLib**: contains Network Integrity libraries (JAR files)

- **lib**: Copy any third party JAR library into this directory.

Switch to the Packager Explorer view, and modify the Java class path to include any JAR files that have been added to this directory.

Select the project, and click F5 to refresh the project in Design Studio to get the modified Java class path affected.

- **model**: contains all Network Integrity models

- **out**: Output directory for compiled Java classes

- **resources**: contains resources related to Network Integrity. This directory is empty by default

- **src**: the Java source directory.
Packaging Cartridges

Packaging the cartridge is the last stage in building a Network Integrity cartridge. The Network Integrity cartridge is packaged as an Integrity ARtifact (IAR) file, which can be deployed to the Network Integrity server through the Cartridge Management Web Service (CMWS) either from Design Studio, or through the Oracle Cartridge Deployer.

The IAR file contains the following build artifacts:

- **IAR root/**
  - `<cartridge-ejb-jar>.jar` - This jar contains manifest.mf file to refer to the jars under cartridgeLib/ `<cartridgeName>`.
  - `oracle.communications.platform.entity.impl.SpecificationDAO`
  - `oracle.communications.platform.entity.impl.CharacteristicSpecUsageDAO`
  - `oracle.communications.platform.entity.impl.CharacteristicSpecificationDAO`
  - `<cartridgeName_A>.mar`
  - `<cartridgeName_B>.mar`
  - `<cartridgeName_N>.mar` - Multiple MAR files if this cartridge is reusing Actions from other cartridges.
  - `<Action_Name_A>_MetaModel.xml`
  - `<Action_Name_B>_MetaModel.xml`
  - `<Action_Name_N>_MetaModel.xml` - Meta Model XML file per Action.

- **/META-INF/**
  - `cartridge.xml`
  - `manifest.xml`

If a Network Integrity cartridge contains only abstract actions, no IAR file is generated. See "Working with Actions" for more information about actions.

Viewing and Solving Problems in the Problems View

The Problems view is the primary source you should refer to for identifying problems with your projects.

The Problems view displays warnings, errors, and informational messages that are generated as you work on projects. For example, if one or more of your saved Java source files contains syntax errors, these errors appear in the Problems view. Similarly, if you make configuration errors while working on specifications, you see error messages in the Problems view.

By default, the problems are logged by severity. You can also group the problems by type. The first column of the view displays an icon that denotes the message type (warning, error, or informational) and the description. The remaining columns display the name of the resource that generated the problem, its path, and its directory location.

You can click an item to open an editor of the affected item so that you can resolve the problem. In code files, the line containing the problem is highlighted.

You can filter the Problems view to display only warnings and errors that are associated with a particular resource or group of resources. See Workbench User Guide in the Design Studio Help for information about applying filters to the Problems view.
Reviewing the Error Log

The Error Log view captures all warnings and error messages logged by actions. Reviewing the Error Log view can help you find errors in your project builds. The underlying log file resides in the metadata subdirectory of your workspace directory.

Error messages that contain a plus sign preceding the message contain other errors. To display these error messages, click the plus sign to expand the view.

To view the details of any error, double-click the message to display the Event Details dialog box.
Deploying and Undeploying Cartridges

Network Integrity cartridges can be directly deployed or undeployed from Design Studio. The deployment or undeployment of Network Integrity cartridges directly from Design Studio should be carried out by Network Integrity cartridge developers only.

Use the Oracle Cartridge Deploer to deploy or undeploy any productized Network Integrity cartridge into a production system.

---

**Note:** Before deploying or undeploying cartridges, ensure that:

- You are logged out of the WebLogic Server Administration Console.
- No one else is deploying or undeploying cartridges on the same server.
- Network Integrity is not running a scan that makes use of the cartridge.

---

This chapter consists of the following topics:

- Creating a Design Studio Environment Project
- Switching to the Studio Environment Perspective
- Deploying a Cartridge
- Undeploying a Cartridge

### Creating a Design Studio Environment Project

Design Studio projects are collections of folders and files that represent the content you are working on. They are used for builds, version management, sharing, and resource organization. Projects map to directories in the file system. When you create a project, you specify a location for it in the file system. Studio uses the files and folders in a project to build a cartridge that you can import into Network Integrity. Each project is a cartridge. See "Building and Packaging Cartridges" for more information. To deploy or undeploy a cartridge from Design Studio, you must first create a Studio Environment Project. When you create a project, you specify its name and location for its corresponding file structure.

To create a Design Studio environment project:

1. Ensure that the Build Automatically setting is enabled. See "Building Cartridges Automatically".
2. Open the Studio Design perspective.

   **Note:** The procedures for creating projects assume that you are working in the Studio Design perspective. For more information about using perspectives, see Design Studio Help.

3. Within the Cartridge view, right-click and select New, then select Environment Project.
   
The New Studio Environment Project wizard appears.

   Figure 21–1 shows an example of a project, named my_integrity_studio_project.

   **Figure 21–1 Creating a New Studio Project**

   ![New Studio Environment Project](image)

   **Studio Project Info**
   
   This wizard creates a new studio project.

   - **Project name:** my_integrity_studio_project
   - **Use default location**
   - **Location:** D:\Network Integrity Workspace
   - **Choose file system:** default

4. In the **Project name** field, enter the name of your project.
   
The project name cannot contain spaces, slashes, or backslashes.

5. Create a location to contain your Network Integrity cartridge project, or verify that the **Use default location** check box is selected.
   
The default location is the workspace directory that contains your Network Integrity cartridge project.

6. Click **Finish**.
   
   Design Studio creates the project and its file structure. The perspective you are using determines how it is displayed. In the Studio Design perspective, the project appears in the Cartridge view, with the Integrity Cartridge editor displaying its properties.

**Creating a Studio Environment**

Having created a Studio Environment Project, you then create the environment. An environment represents a connection to a particular server.

To create a Studio environment:
1. Right-click the `my_integrity_studio_project` project, select New, select Environment, and then select Studio Environment.

The Studio Environment Wizard appears. Figure 21–2 shows an example of an environment, named my_server.

**Figure 21–2   Creating a New Studio Environment**

![Studio Environment Wizard](image)

Project: `my_integrity_studio_project`
Type: Studio Environment

Name: `my_server`
Folder: 

2. In the Name field, enter the name of your environment. The name should incorporate the server name.

The environment name cannot contain spaces, slashes, or backslashes.

3. Create a location to contain your environment.

4. Click Finish.

Design Studio creates the environment and its file structure. The perspective you are using determines how it is displayed. In the Studio Design perspective, the environment appears in the Cartridge view, with the Integrity Cartridge editor displaying its properties.

Figure 21–3 shows the connection details for the environment.

**Figure 21–3   Configuring the Studio Environment Connection information**

![Environment Connection Information](image)
5. In the Cartridge view for the environment, click the **Connection** tab.
   a. (Optional) In the **Description** field, provide a description of your environment.
   b. In the **Address** field, specify the Cartridge Management Web Service (CMWS) URL.

   **Note:** If SSL is used, ensure that `https` is specified in this field.

6. (Optional) In the Cartridge view for the environment, click the **SSL** tab. This step is required only if Hypertext Transfer Protocol Secure (HTTPS) is used.
   a. (Optional) In the **Description** field, provide a description of your environment.
   b. In the **Keystore** field, browse to the location of the keystore file on the local hard drive.

   The keystore file must be transferred from the WebLogic server to the location where Design Studio is running. If you select the WebLogic demo keystore, you must add the host name instead of the IP address in the **Address** field under the **Connection** tab.

   This is because the WebLogic demo keystore was generated against the host name, not the IP address, during Network Integrity installation.

7. In the Cartridge view for the environment, click the **Properties** tab.
   Specify the properties for cartridge management so that Design Studio can successfully deploy and undeploy Network Integrity cartridges:
   a. Expand the **Network Integrity Cartridge** folder, and then expand the **Cartridge Management** folder.
   b. Configure the following properties:
      - `wladmin.host.name`: The host name or IP address where the Oracle WebLogic Admin Server is running
      - `wladmin.host.port`: The port number on which the Oracle WebLogic Admin Server is running
      - `wladmin.server.name`: The Oracle WebLogic Admin Server name.

8. Click **Save** to save the Studio environment project. Design Studio automatically builds the environment project. When the build is completed, there should be no warnings or errors from this environment project. You can now deploy and undeploy Network Integrity cartridges.

---

**Switching to the Studio Environment Perspective**

Before deploying or undeploying cartridges from Design Studio, you must open the Studio Environment perspective.

To open the Studio Environment perspective, do one of the following:

- From the **Window** menu, select **Open Perspective**, then select **Studio Environment**
- From the **Studio** menu, select **Other**

The Open Perspective dialog appears
Select **Studio Environment**
Click **OK**
- From the **Studio** menu, select **Show Environment Perspective**.

The Studio Environment perspective displays two views:
- Environment
- Cartridge Management

If they do not open, you can manually open these two views.

To open the Studio Environment views, do one of the following:
- From the **Window** menu, select **Show View**, then select **Environment** and **Cartridge Management**
- From the **Window** menu, select **Other**
  
  The Show View dialog appears
  
  Expand the **Oracle Design Studio** folder
  
  Select **Environment** and **Cartridge Management**

**Connecting to the Studio Environment**

Having opened the Environment and Cartridge Management views in the Studio Environment perspective, you can connect to the environment that has been created.

1. Within the **Environment** view, right-click **my_server**, then select **Test Connection**.
   
   The Test Environment Connection dialog appears.
   
   **Figure 21–4** shows an example of a test connection.

2. In the **User Name** field, enter the Cartridge Management Web Service (CMWS) username.
3. In the **Password** field, enter the password associated with the CMWS username.
4. Click **OK** to test the connection.
Deploying a Cartridge

Before deploying a Network Integrity cartridge in Design Studio, you must first build the cartridge and ensure that there is no error marker associated with it. See "Building Cartridges Automatically".

To deploy a cartridge:

1. Open the Studio Environment perspective. See "Switching to the Studio Environment Perspective".
2. If Design Studio is not connected to the target environment, make the connection by following the steps outlined in "Connecting to the Studio Environment".
3. From the Cartridge Management view, select the cartridge that you want to deploy, and click Deploy to deploy this cartridge to the target environment.

   a. (Optional) Click Run in Background to run the deployment in the background.
   b. (Optional) Check the Always run in background box to run all future cartridge deployments in the background.
   c. (Optional) Click Details to view the environment connection properties. See "Connecting to the Studio Environment".

When deployment is completed without error, a Deploy Completed confirmation dialog appears.

4. Click OK.

Design Studio deploys the cartridge, and lists the newly deployed cartridge with version and timestamp in the Deployed Versions table. The Console view opens to provide extra information regarding the deployment. Error messages are displayed if any exist.

Note: The Cartridge Management view displays a list of cartridges available in the current workspace. However, a cartridge that contains only abstract actions does not appear in this list.

Figure 21–5 shows a deployed cartridge called Address_Handlers.
Redeploying a Cartridge

You can redeploy a Network Integrity cartridge using Design Studio, only if the version of the redeployed cartridge (build number) is equal to, or greater than, the version of the deployed cartridge. For example, my_cartridge is already deployed with a build number of 28 (b28). If my_cartridge is up-versioned to b30, you can deploy it without undeploying my_cartridge (b28) and deploying it again.

Redeployment removes the deployed cartridge and deploys the new cartridge instead. Network Integrity does not allow more than one version of the same cartridge to be deployed at the same time.

Undeploying a Cartridge

When a cartridge is undeployed, Network Integrity removes all the scan configurations and scan results associated with the cartridge and all the specifications associated with the cartridge (except those specifications still in use by other cartridges).

If a cartridge has a dependency on other deployed cartridges, the cartridge cannot be undeployed. For example, you cannot undeploy the Address_Handlers cartridge if the cartridges using Address_Handlers are still deployed in Network Integrity. You must undeploy all dependent cartridges from Network Integrity before Address_Handlers can be undeployed.

The Network Integrity Cartridge Management Web Service (CMWS) Adapter automatically carries out dependency checks at deployment or undeployment time and returns error messages if deployment or undeployment cannot be carried out.

To undeploy a cartridge:

1. Open the Studio Environment perspective. See "Switching to the Studio Environment Perspective".
2. If Design Studio is not connected to the target environment, make the connection by following the steps outlined in "Connecting to the Studio Environment".
3. From the Deployed Versions table in the Cartridge Management view, select the cartridge that you want to undeploy, and click Undeploy to undeploy this cartridge from the target environment.

A Confirm to Undeploy dialog appears.
4. Click **OK** to confirm the undeployment.
   
   Design Studio undeploys the cartridge, and removes the cartridge from the Deployed Versions table. The Console view opens to provide extra information regarding the undeployment. Error messages are displayed if any exist.

5. (Optional) You can click **Cancel** to attempt to cancel this undeployment. If the undeployment request has already been sent to CMWS, this undeployment cannot be cancelled.

When a cartridge is undeployed, Network Integrity removes all of the scan configurations and scan results associated with this cartridge and all of the specifications (except those still in use by other cartridges) associated with this cartridge.

If a cartridge has a dependency with other deployed cartridges, this cartridge cannot be undeployed. For example, you cannot undeploy *Address Handlers* if the cartridges that are using the Address Handler are still deployed on Network Integrity.

You must undeploy all dependent cartridges from Network Integrity before the Address Handlers can be undeployed. The Network Integrity CMWS Adapter automatically carries out the dependency check at deployment or undeployment time.
This chapter provides information about debugging and testing cartridges in Oracle Communications Network Integrity.

**Starting the WebLogic Server in Test Mode**

To debug a deployed Network Integrity cartridge, start the WebLogic Managed Server in debug mode (not the Admin Server).

Use the following procedure to start the WebLogic Managed Server in debug mode:

1. Stop both the Admin Server and Managed Server if they are still running.
2. Go to directory `<WEBLOGIC_HOME>/user_projects/domains/<DOMAIN>/bin`.
3. Copy the existing `startWebLogic.sh` script to a new script file, `startWebLogic_Debug.sh`.
4. Use a text editor to open `startWebLogic_Debug.sh`.
5. After the line `${JAVA_HOME}/bin/java ${JAVA_VM} -version`, add the following two lines:
   ```
   echo "Launching Java with debug port: 10171"
   JAVA_OPTIONS="-Xdebug -Djava.compiler=NONE -Xnoagent -Xrunjdwp:transport=dt_socket,server=y,address=10171,suspend=n $JAVA_OPTIONS"
   ```
   The debug port does not have to be 10171 if the port specified is available.
6. Save this change.
7. Copy the existing `startManagedWebLogic.sh` script to a new script file, `startManagedWebLogic_Debug.sh`.
8. Use a text editor to open `startManagedWebLogic_Debug.sh`.
9. Find the two lines that are referring to `startWebLogic.sh`.
10. Replace `startWebLogic.sh` with `startWeblogic_Debug.sh`. This change is to start the WebLogic Managed Server in debug mode by invoking the `startWebLogic_Debug.sh` script.
11. Save this change.
12. Start the Admin Server by running the usual start-up script, `startWebLogic.sh`.
13. Start the Managed Server in debug mode by running the new script, `startManagedWebLogic_Debug.sh`. 


Configuring Remote Debugger in Design Studio

The Managed Server is now in debug mode. The next step is to configure the debugger in Eclipse to start remote-debugging the Network Integrity cartridges.

1. From the Design Studio main menu, select Run then Debug Configurations, then open the Debug Configurations dialog to switch Design Studio to the Java perspective.
2. From the left panel, select Remote Java Application.
3. Click New to create a remote Java application debug configuration.
4. Enter a name for this new debug configuration.
5. In the Connect tab, click the Browse.
6. Select an available project that contains the cartridge that to debug.
7. Ensure that the default setting for Connection Type is Standard (Socket Attach).
8. Enter the host IP address where the Network Integrity system (WebLogic Managed Server) is running.
9. Enter the debug port, which should match the debug port entered in "Starting the WebLogic Server in Test Mode".
10. Keep the default settings for the rest of the tab.
11. Click Apply to save this new remote Java application debug configuration.

Now the developer can start to debug the Network Integrity cartridge (which should be already deployed on the Network Integrity system) from Design Studio by picking up the debug configuration just created. There is no difference from debugging a normal local Java application in Eclipse. We can put a break point in the cartridge Java source and start debugging from there. Please refer to Eclipse on-line help contents, Java development user guide then Getting Started then Basic tutorial then Debugging your programs, on how to debug a Java program in Eclipse.
Sealing and Unsealing Cartridges

Network Integrity production cartridges are distributed as sealed cartridges. Unsealing of Oracle Communications Network Integrity production cartridges violates the license, support, and maintenance agreements with Oracle.

*Getting Started with Design Studio* explains the differences between sealed and unsealed cartridges, and describes the procedures used to seal or unseal a cartridge.
This chapter provides an overview of exporting and importing Network Integrity cartridges.

Cartridge projects can be exported to archive files. This allows the cartridge projects to be distributed as a single or a set of archive files, rather than as the many files of a cartridge project. Once a project or projects is exported to an archive file, the archive file can be distributed and then imported into a different Studio or Eclipse workspace.

Before exporting a cartridge project, you should decide whether you want to include your source code in the archive file. Cartridges can be extended without distributing source code. However, if you want to allow the user to modify the actual distributed cartridge, then you must distribute the source code.

Cartridges can also be exported in both sealed and unsealed states. It is recommended that if you are distributing a cartridge without source code, you should seal the cartridge before exporting it. This prevents the user from changing the cartridge model and therefore breaking the cartridge.

To understand the differences between sealed and unsealed cartridges, and the procedures for sealing or unsealing a cartridge, see Getting Started with Design Studio.

Network Integrity production cartridges are distributed as sealed cartridges. Unsealing Network Integrity production cartridges violates the license, support, and maintenance agreements with Oracle.

See the following:

- Exporting a Cartridge With Source Code
- Exporting a Cartridge Without Source Code
- Importing a Cartridge

### Exporting a Cartridge With Source Code

To export a cartridge containing source code:

1. From the Studio File menu, select Export.
   
   The Export Select dialog appears.

2. From the list of export destinations, expand the General node and select Archive File.

3. Click Next.
   
   The Export Archive file dialog appears.

4. Enter a destination archive file:
a. Select the projects that you want to include in the archive.

b. Specify the name and location of the archive file.

c. In the Options section, accept the defaults.

d. Click Finish to create an archive file containing the exported projects at the specified location.

Figure 24–1 displays cartridge resources being archived.

Figure 24–1 Archiving Cartridge Files

Exporting a Cartridge Without Source Code

Before exporting a cartridge project without source code, the project’s classpath must be modified.

See the following:

- Modifying the Classpath
- Exporting the Cartridge

Modifying the Classpath

To modify the classpath:

1. Open the Navigator view.

2. Use the Navigator view to rename the projects output directory out, to classes.
3. From the Studio Window menu, select Show View, and select the Package Explorer view.

4. Select the project.

5. From the Studio Project menu, select Properties.
   The Properties dialog appears.

6. From the list of properties, expand the Java Build Path node and select Source.
   A list of source folders on the build path is displayed.

7. Remove the source directories that are part of the classpath:
   a. Select the source folders on the build path.
   b. Click Remove.

   Figure 24–2 displays source projects being removed.

**Figure 24–2 Removing Source Projects**

8. Select the Libraries tab, and click Add Class Folder to add the class folder classes, from step 2, to the classpath.

   Figure 24–3 shows how the class folder is added to the classpath.
9. Select the **Order and Export** tab, and check the box corresponding to the class folder **classes**.

   **Figure 24–4** shows how the class folder is exported.

10. Click **OK** to complete the modification of the project classpath.

    After changing the classpath, if you wish to continue development on the cartridge, you should restore the classpath to its original configuration.
Exporting the Cartridge

Now that you’ve modified the classpath, you can export the cartridge:

1. From the Studio File menu, select Export.
   The Export Select dialog appears.
2. From the list of export destinations, expand the General node and select Archive File.
3. Click Next.
   The Export Archive file dialog appears.
4. Enter a destination archive file:
   a. Select the projects that you want to include in the archive.
   b. For the projects for which you are not including source code, expand the project tree and deselect the source directories which you previously removed from the classpath.
   c. Specify the name and location of the archive file.
   d. In the Options section, accept the defaults.
   e. Click Finish to create an archive file containing the exported projects at the specified location.

Importing a Cartridge

In addition to creating your own cartridges, you can import existing cartridges into Studio. When you import a cartridge, it becomes a project in the current workspace.

If you import a cartridge from a technology pack or a cartridge that has been protected by its creator, it may be sealed. Sealed cartridges cannot be modified without first being unsealed.

WARNING: While it is possible to unseal a cartridge from a technology pack, you should not do so. A technology pack is supported by Oracle only if it remains sealed. You can extend a technology pack without unsealing its cartridges.

Cartridges with dependencies display errors when you import them without the required cartridges already being present in the workspace. For example, if you import a cartridge that requires a base cartridge, you see errors. After all the cartridges have been imported, you can clean all the projects to remove the errors.

To import a cartridge project from an archive file:

1. From the Studio File menu, select Import.
   The Import Select dialog appears.
2. From the list of import sources, expand the General node and select Existing Projects into Workspace.
3. Click Next.
   The Import Projects dialog appears.
4. Select the cartridge project:
a. Click **Select archive file**.

b. Specify the name and location of the archive file. A list of the projects in the archive file is displayed.

c. Select the required projects.

d. Click **Finish** to import the projects into your workspace.

*Figure 24–5 shows how the project is imported.*
This chapter provides information on working with source control systems and Design Studio in Oracle Communications Network Integrity.

Working with Source Control

When developing cartridges with Design Studio for Network Integrity, developers may store their work in various source control systems. The eclipse platform, upon which Design Studio is base, provides support for integrating with source control systems. Plugins are available for most common source control systems. The exact behavior of Design Studio when used in an environment where the files are backed by a source control system depends on the source control system and the source control Team plugin that the developer is using.

This section describes which files must be source controlled and which files must be writable to continue working.

Table 25–1 describes the structure of the directories and the files in a Design Studio for Network Integrity project and recommends how they should be handled with respect to a source control system.

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Description</th>
<th>Source Control Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectDir/</td>
<td>Project’s top level directory.</td>
<td>Under source control. All files directly under this directory must be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>source controlled.</td>
</tr>
<tr>
<td>ProjectDir/cartridgeBin/</td>
<td>Cartridge bin directory is where the deployable IAR files are located.</td>
<td>This directory should be source controlled but the contents should not.</td>
</tr>
<tr>
<td>ProjectDir/cartridgeBuild/</td>
<td>Cartridge build directory contains files which are outputs of the cartridge build process.</td>
<td>This directory should be source controlled but the contents should not.</td>
</tr>
<tr>
<td>ProjectDir/dataDictionary/</td>
<td>This directory contains the files where the data dictionary information is stored.</td>
<td>This directory and its contents should be source controlled.</td>
</tr>
<tr>
<td>ProjectDir/doc/</td>
<td>This directory contains documentation files.</td>
<td>This directory and its contents should be source controlled.</td>
</tr>
<tr>
<td>ProjectDir/generated/</td>
<td>This directory contains generated artifacts of the build process.</td>
<td>This directory should be source controlled. Except for the src sub-directory, the contents of this directory should not be source controlled.</td>
</tr>
<tr>
<td>ProjectDir/generated/src/</td>
<td>This directory contains generated artifacts of the build process.</td>
<td>This directory should be source controlled, but it contents should not.</td>
</tr>
</tbody>
</table>
### Table 25–1 (Cont.) Source Control Handling for Various Files and Directories

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Description</th>
<th>Source Control Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectDir/integrityLib/</td>
<td>This directory contains jars that are part of the Network Integrity server Enterprise Archive (EAR). These jars are in the project’s classpath.</td>
<td>This directory should be source controlled. The files in this directory should not be source controlled.</td>
</tr>
<tr>
<td>ProjectDir/integrityLib/packages</td>
<td>This directory contains jars that are created by Design Studio for Network Integrity and which are packaged into the cartridge IAR file. The jars are added to the Network Integrity EAR when the cartridge is deployed. These jars are in the project’s classpath.</td>
<td>This directory should be source controlled. The files in this directory should not be source controlled.</td>
</tr>
<tr>
<td>ProjectDir/lib/</td>
<td>This directory contains jars and other files that are not part of the Network Integrity server EAR. Some of these files are part of the project classpath.</td>
<td>This directory should be source controlled. The mds.mar file is output to this directory. The mds.mar file should not be source controlled. The user may also want to source control other files in this directory.</td>
</tr>
</tbody>
</table>
| ProjectDir/mdsArtifacts/ | This directory contains files that are both input and outputs of the UI Hints infrastructure. | This directory should be source controlled. The following files under this directory should also be source controlled:  
- MDSAvailablePagePanels.xml  
- MDSAvailablePagePanels.xsd  
- MDSMetaData.xml  
The remaining files in this directory should not be source controlled. |
| ProjectDir/model/ | This directory contains files that are used to persist the information about Cartridges, Actions, Processors, Model Collections and Address Handlers. | This directory and its contents should be source controlled. |
| ProjectDir/out/ | This directory contains output classes. | This directory should not be source controlled. |
| ProjectDir/resources/ | This directory is not used. | This directory does not need to be source controlled. |
| ProjectDir/src/ | This directory contains the user supplied code for the cartridge. | This directory and its contents should be source controlled. |

Design Studio for Network Integrity assumes that all files and directories of a cartridge project are writable. Some source control systems and team plug-ins automatically manage the files and directories to make them writable as the software needs to write to them. If this is not the case for your chosen source control/Team plug-in combination, then you should manually ensure that this is the case before working with a source controlled project.
This chapter provides information about the Web Service API Oracle Communications Network Integrity.

Web Service Overview

The Network Integrity Web Services API enables Oracle Communications products and third party applications to interact with Network Integrity and reduces integration complexity by providing a standards-based interface. With the API, clients can externally manage Network Integrity through Web Services APIs.

At a high-level the web service supports:

- Configuring all types of Scans
- Running Discovery and Reconciliation Scans
- Retrieving scan results including any found discrepancies
- Initiating corrective actions such as reconciling discrepancies in Inventory systems.

Most all operations that can be done in the Web UI are possible using the Web Service API. One operation that is currently not possible is to create or update the Import System configured in the Web UI. This is a one-time setup that must be done in the Web UI and cannot be done using the Web Service API.

The Web Service API is standards based using JAX-WS over HTTP.

Security

The Network Integrity Web Service API uses the same security as the Network Integrity Web User Interface. So any user able to login into the web UI can also use the Web Service API. This is assigned using the NetworkIntegrityRole.

Model Based

The Network Integrity Web Service API operates on the Network Integrity Model. Knowledge of the entities, attributes and relationships in the Network Integrity model is essential for using the Web Service API.

For Network Integrity entity, attribute and relationship names, refer to Network Integrity Information Model Reference.

For cartridge entity, parameter, and relationship names and descriptions, refer to your cartridge documentation.
Concurrency With UI and other Web Service Clients

Web Service API operations take immediate effect in the system and therefore there is potential for collisions with users working in the Web UI. If the Web Service API operation collides with an update that another user has done in the Web UI or another Web Service client, then an error is returned to a client. For example, if a Web Service client deletes a DisConfig (Scan) while a user is editing the same scan in the Web UI, the Web UI user receives an error when that user attempt to save changes. If two clients (Web Service client or Web UI user) are trying to update/delete the same entity, the last client to commit changes receives the error.

All Operations

Table 26–1 describes the DisConfig operations. See Network Integrity Information Model Reference for further information on this entity.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDisConfig</td>
<td>This operation creates a new Scan in the system (DisConfig is equivalent to Scan in the Network Integrity Web UI).</td>
</tr>
<tr>
<td>deleteDisConfig</td>
<td>This operation deletes a Scan from the system. All results and discrepancies produced by this scan are deleted as well. A Fault is returned if the delete fails.</td>
</tr>
<tr>
<td></td>
<td>This delete operation returns a Fault if the scan to be deleted has discrepancies in the Received or the Submitted state. Add <a href="">v1:forceDelete</a>YES&lt;/v1:forceDelete&gt; to the delete request to force the scan to delete and bypass this particular Fault.</td>
</tr>
<tr>
<td>findDisConfig</td>
<td>This operation finds Scans in the system based on search criteria provided in the request. Full Scan data is returned but client applications can limit the amount of data returned, or support paging, by providing a fromRange and toRange in the request. A Fault with a faultstring is returned if the find fails.</td>
</tr>
<tr>
<td>getDisConfig</td>
<td>This operation gets the details about a Scan. It requires the DisConfig entity ID to be passed in the request, and returns the full details of the Scan including UI Parameters, Scope Addresses, and Schedule information in the response, if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>updateDisConfig</td>
<td>This operation updates a Scan in the system. All the values for the Scan are required in the request, not just the values changing, therefore the client application should perform a ‘get’ operation and update the required values for the update operation. A Fault with a faultstring is returned if the update fails.</td>
</tr>
</tbody>
</table>

Table 26–2 describes the DisScanRun operations. See Network Integrity Information Model Reference for further information on this entity.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findDisScanRun</td>
<td>This operation finds Scan Results in the system based on search criteria provided in the request (DisScanRun is equivalent to Scan Results in the Network Integrity Web UI). Full Scan Result data is returned but client applications can limit the amount of data returned, or support paging, by providing a fromRange and toRange in the request. A Fault with a faultstring is returned if the find fails.</td>
</tr>
<tr>
<td>deleteDisScanRun</td>
<td>This operation deletes Scan Results from the system. All results and discrepancies attached to the scan results are deleted as well. A Fault with a faultstring is returned if the delete fails.</td>
</tr>
<tr>
<td>getDisScanRun</td>
<td>This operation gets all the details about an instance of Scan Results. The operation requires the Discrepancy entity id to be passed in the request, and returns the full details of the Discrepancy including references to the compare and reference Oracle Communications Information Model entities which the discrepancy was found on. If not found, a Fault is the response.</td>
</tr>
</tbody>
</table>
Table 26–3 describes the DisBlackoutSchedule operations. See Network Integrity Information Model Reference for further information on this entity.

### Table 26–3  DisBlackoutSchedule Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDisBlackoutSchedule</td>
<td>This operation creates a new Blackout Schedule in the system. A recurrence rule, duration, and start time are required in the request. The Blackout Schedule can be assigned to Scan Configurations on creation, or they can be associated later with an update operation.</td>
</tr>
<tr>
<td>deleteDisBlackoutSchedule</td>
<td>This operation deletes a Blackout Schedule in the system. If any Scans are associated with the Blackout Schedule then the associations are removed as well. A Fault with a faultstring is returned if the delete fails.</td>
</tr>
<tr>
<td>getAllDisBlackoutSchedule</td>
<td>This operation returns the full details of all the Blackout Schedules in the system. An empty response is returned if no Blackout Schedules exist in the system.</td>
</tr>
<tr>
<td>getDisBlackoutSchedule</td>
<td>This operation requires the Blackout Schedule entity id to be passed in the request, and returns the full details of the Blackout Schedule in the response if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>updateDisBlackoutSchedule</td>
<td>This operation updates a Blackout Schedule in the system. All the values for the Blackout Schedule are required in the request, not just the values changing. A Fault with a faultstring is returned if the update fails.</td>
</tr>
</tbody>
</table>

Table 26–4 describes the DisTag operations. See Network Integrity Information Model Reference for further information on this entity.

### Table 26–4  DisTag Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDisTag</td>
<td>This operation creates a new Tag, a name for the tag is required. The parent tag entity id can be provided in the creation or can be add after in an update request. A Fault with a faultstring is returned if the delete fails.</td>
</tr>
<tr>
<td>deleteDisTag</td>
<td>This operation deletes the specified Tag and all child Tags. The entity id of the Tag to be deleted is required. If any Scans are associated with the Tag then the associations are removed as well. A Fault with a faultstring is returned if the delete fails.</td>
</tr>
<tr>
<td>getDisTag</td>
<td>This operation requires the Tag entity id to be passed in the request, and returns the full details of the Tag including all child tags in the response, if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getAllRootDisTags</td>
<td>This operation returns the full details of all the Tags configured in the system. The root Tags returned also include the details of children Tag entities. A Fault with a faultstring is returned if an error occurs.</td>
</tr>
<tr>
<td>updateDisTag</td>
<td>This operation updates a Tag, an entity id and name for the tag is required. All the values for the Blackout Schedule are required in the request, not just the values that are changing. Modifications to the hierarchy must be performed on the child Tag, for example, to make a child Tag a root Tag call the update operation with no parent Tags specified. A Fault with a faultstring is returned if the update fails.</td>
</tr>
</tbody>
</table>

Table 26–5 describes the DisDiscrepancy operations. See Network Integrity Information Model Reference for further information on this entity.
All Operations

Table 26–5  DisDiscrepancy Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findDisDiscrepancy</td>
<td>This operation finds Discrepancies in the system based on search criteria provided in the request. The search criteria available in the Web Service API operation is the same as the criteria available in the Network Integrity UI (DisScanRun is equivalent to Scan Results in the Network Integrity Web UI). Full Discrepancy data is returned but client applications can limit the amount of data returned, or support paging, by providing a fromRange and toRange in the request. A Fault with a faultstring is returned if the find fails.</td>
</tr>
<tr>
<td>getDisDiscrepancy</td>
<td>This operation gets all the details about a Discrepancy. The operation requires the Discrepancy entity id to be passed in the request, and returns the full details of the Discrepancy including references to the compare and reference Information Model entities which the discrepancy was found on. If not found, a Fault is the response.</td>
</tr>
</tbody>
</table>
| updateDisDiscrepancy     | This operation updates a Discrepancy in the system. All the values for the DisDiscrepancy are required in the request, not just the values changing. The valid values of Status are  
  - DISCREPANCY_OPENED  
  - DISCREPANCY_IGNORED  
  - OPERATION_IDENTIFIED  
  - OPERATION_SUBMITTED  
  - OPERATION_RECEIVED  
  - OPERATION_NOT_IMPLEMENTED  
  - OPERATION_PROCESSED  
  - OPERATION_FAILED  
  The operation value is equivalent to Resolution Action value in the Network Integrity UI and the valid values is dependent on what Discrepancy Resolution are currently installed in the system. A Fault with a faultstring is returned if the update fails. |

Table 26–6 describes the DisPlugin operations. See Network Integrity Information Model Reference for further information on this entity.

Table 26–6  DisPlugin Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllDisAssimilationPlugin</td>
<td>This operation returns details about all Assimilation Plugins deployed in the system (AssimilationPlugin is equivalent to Assimilation/Scan Action in the Network Integrity UI).</td>
</tr>
<tr>
<td>getAllDisInventoryImportPlugin</td>
<td>This operation returns details about all Import Plugins deployed in the system (InventoryImportPlugin is equivalent to Import/Scan Action in the Network Integrity UI).</td>
</tr>
<tr>
<td>getAllDisNetworkDiscoveryPlugin</td>
<td>This operation returns details about all Discovery Plugins deployed in the system (NetworkDiscoveryPlugin is equivalent to Discovery/Scan Action in the Network Integrity UI).</td>
</tr>
<tr>
<td>getAllDisDiscrepancyDetectionPlugin</td>
<td>This operation returns details about all Discrepancy Detection Plugins deployed in the system (Discrepancy Detection Plugin is equivalent to a Discrepancy Detection Action)</td>
</tr>
<tr>
<td>getAllDisDiscrepancyResolutionPlugin</td>
<td>This operation returns details about all Discrepancy Resolution Plugins deployed in the system (Discrepancy Resolution Plugin is equivalent to a Discrepancy Resolution Action)</td>
</tr>
</tbody>
</table>
Table 26–7 describes the DefaultDisInventoryConfig operations.

**Table 26–7 DefaultDisInventoryConfig**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDefaultDisInventoryConfig</td>
<td>This operation returns the inventory system configured in the Network Integrity system. This is the inventory system configuration that is entered in the “Manage Import System” task of the Network Integrity Web UI. The Import System cannot be created or updated using the Web Service API, it must be done using the Network Integrity Web UI.</td>
</tr>
</tbody>
</table>
Table 26–8 Special Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startScan</td>
<td>This operation starts a scan. The response returns a reference to the Scan Result entity so that the client application can monitor the progress of the scan. (DisScanRun is equivalent to the Scan Results in the Network Integrity UI). If the Scan is already running or in the process of stopping then the startScan operation fails. If the scan could not be started, a Fault with a reason is the response.</td>
</tr>
<tr>
<td>stopScan</td>
<td>This operation stops a scan that is running. The scan is transitioned to a STOPPING state immediately and then transition to STOPPED when actually ended. If the Scan is not currently running, this call is a no-op. If the scan could not be transitioned to Stopping, a Fault with a reason is the response.</td>
</tr>
<tr>
<td>submitDisDiscrepancy</td>
<td>This operation submits the list of discrepancies provided in the request for resolution processing. The status of the discrepancies must be ‘OPERATION_IDENTIFIED’ to submit them, otherwise a Fault is returned. A Fault with a faultstring is returned if the operation fails.</td>
</tr>
<tr>
<td>ResolutionOperations</td>
<td></td>
</tr>
<tr>
<td>getLatestScanStatus</td>
<td>This operation returns the scan status for the most recent execution of a scan. This operation is more efficient than getDisScanRun and therefore is more appropriate for client applications that are monitoring the status of a scan (DisConfig is equivalent to Scan in the Network Integrity Web UI). A Fault with a faultstring is returned if the operation fails.</td>
</tr>
</tbody>
</table>

Table 26–9 describes the Information Model entity operations. Information Model entities are described in Oracle Communications Information Model Reference and Network Integrity Information Model Reference.

Table 26–9 Information Model Entity Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getRootEntity</td>
<td>This operation gets all the details about a discovered, imported, or assimilated root Information Model entity. The root entity id for the request is obtained from either a getDisScanRun operation response or findDisScanRun operation response. The id is found in the 'rootEntityRefsRef' element in the result groups. Multiple ids can be passed in the request. The response entity can be many different types depending on what the cartridge persisted in the result group. An example root entity type is Physical Device or Logical Device, but other Information Model types are possible. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getResultEntity</td>
<td>A generic operation to get any type of Information Model entity given an entityId and the entity type. Multiple entities can be retrieved in a single request. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getSpecification</td>
<td>This operation gets all the details about specification deployed in the system. Most Information Model entities support specifications which is blueprint for what characteristics are supported, among other things. All the characteristics defined in this specification are returned. Specifications are deployed to the system when cartridges containing them are deployed. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getLogicalDevice</td>
<td>This operation requires the LogicalDevice entity id to be passed in the request, and returns the full details of the LogicalDevice if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getDeviceInterface</td>
<td>This operation requires the DeviceInterface entity id to be passed in the request, and returns the full details of the DeviceInterface if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getMediaInterface</td>
<td>This operation requires the MediaInterface entity id to be passed in the request, and returns the full details of the MediaInterface if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getLogicalDeviceAccount</td>
<td>This operation requires the LogicalDeviceAccount entity id to be passed in the request, and returns the full details of the LogicalDeviceAccount if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getPhysicalDevice</td>
<td>This operation requires the PhysicalDevice entity id to be passed in the request, and returns the full details of the PhysicalDevice if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getEquipment</td>
<td>This operation requires the Equipment entity id to be passed in the request, and returns the full details of the Equipment if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getEquipmentHolder</td>
<td>This operation requires the EquipmentHolder entity id to be passed in the request, and returns the full details of the EquipmentHolder if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPhysicalPort</td>
<td>This operation requires the PhysicalPort entity id to be passed in the request, and returns the full details of the PhysicalPort if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPhysicalConnector</td>
<td>This operation requires the PhysicalConnector entity id to be passed in the request, and returns the full details of the PhysicalConnector if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getCustomObject</td>
<td>This operation requires the CustomObject entity id to be passed in the request, and returns the full details of the CustomObject if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getCustomNetworkAddress</td>
<td>This operation requires the CustomNetworkAddress entity id to be passed in the request, and returns the full details of the CustomNetworkAddress if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getTelephoneNumber</td>
<td>This operation requires the TelephoneNumber entity id to be passed in the request, and returns the full details of the TelephoneNumber if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getInventoryGroup</td>
<td>This operation requires the InventoryGroup entity id to be passed in the request, and returns the full details of the InventoryGroup if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getService</td>
<td>This operation requires the Service entity id to be passed in the request, and returns the full details of the Service if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getNetwork</td>
<td>This operation requires the Network entity id to be passed in the request, and returns the full details of the Network if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getNetworkNode</td>
<td>This operation requires the NetworkNode entity id to be passed in the request, and returns the full details of the NetworkNode if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getNetworkEdge</td>
<td>This operation requires the NetworkEdge entity id to be passed in the request, and returns the full details of the NetworkEdge if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPipe</td>
<td>This operation requires the Pipe entity id to be passed in the request, and returns the full details of the Pipe if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPipeTerminationPoint</td>
<td>This operation requires the PipeTerminationPoint entity id to be passed in the request, and returns the full details of the PipeTerminationPoint if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPipeDirectionality</td>
<td>This operation requires the PipeDirectionality entity id to be passed in the request, and returns the full details of the PipeDirectionality if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getTrailPath</td>
<td>This operation requires the TrailPath entity id to be passed in the request, and returns the full details of the TrailPath if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getGeographicPlace</td>
<td>This operation requires the GeographicPlace entity id to be passed in the request, and returns the full details of the GeographicPlace if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getGeographicAddress</td>
<td>This operation requires the GeographicAddress entity id to be passed in the request, and returns the full details of the GeographicAddress if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getGeographicAddressRange</td>
<td>This operation requires the GeographicAddressRange entity id to be passed in the request, and returns the full details of the GeographicAddressRange if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getGeographicLocation</td>
<td>This operation requires the GeographicLocation entity id to be passed in the request, and returns the full details of the GeographicLocation if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getGeographicSite</td>
<td>This operation requires the GeographicSite entity id to be passed in the request, and returns the full details of the GeographicSite if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getNetworkNodeRole</td>
<td>This operation requires the NetworkNodeRole entity id to be passed in the request, and returns the full details of the NetworkNodeRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPhysicalConnectorRole</td>
<td>This operation requires the PhysicalConnectorRole entity id to be passed in the request, and returns the full details of the PhysicalConnectorRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPipeRole</td>
<td>This operation requires the PipeRole entity id to be passed in the request, and returns the full details of the PipeRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPhysicalPortRole</td>
<td>This operation requires the PhysicalPortRole entity id to be passed in the request, and returns the full details of the PhysicalPortRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getDeviceInterfaceRole</td>
<td>This operation requires the DeviceInterfaceRole entity id to be passed in the request, and returns the full details of the DeviceInterfaceRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getLogicalDeviceRole</td>
<td>This operation requires the LogicalDeviceRole entity id to be passed in the request, and returns the full details of the LogicalDeviceRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getCustomObjectRole</td>
<td>This operation requires the CustomObjectRole entity id to be passed in the request, and returns the full details of the CustomObjectRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPhysicalDeviceRole</td>
<td>This operation requires the PhysicalDeviceRole entity id to be passed in the request, and returns the full details of the PhysicalDeviceRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getEquipmentRole</td>
<td>This operation requires the EquipmentRole entity id to be passed in the request, and returns the full details of the EquipmentRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getNetworkEdgeRole</td>
<td>This operation requires the NetworkEdgeRole entity id to be passed in the request, and returns the full details of the NetworkEdgeRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getPlaceRole</td>
<td>This operation requires the PlaceRole entity id to be passed in the request, and returns the full details of the PlaceRole if found. If not found, a Fault is the response.</td>
</tr>
<tr>
<td>getNetworkRole</td>
<td>This operation requires the NetworkRole entity id to be passed in the request, and returns the full details of the NetworkRole if found. If not found, a Fault is the response.</td>
</tr>
</tbody>
</table>
Create, Read, Update, Delete Pattern

Most of the operations defined in the Network Integrity Web Service API follow a pattern.

The operation patterns are:

- Create
- Get
- Get All
- Delete
- Update
- Find

The following sections describe each pattern and the common attributes to each. An example request and response is given for each pattern.

There are a few Web Service API operations that do not follow the pattern. There are dedicated sections with examples for each of these.

Create Operation

The create operation inserts a new entity into the system. For example, the createDisBlackoutSchedule operation creates a new Blackout Schedule in the system.

If successful, the changes are immediately available in the system and can be viewed in the Web UI.

The request for the Create operation is named create<EntityType>Request. The request contains the full details of the new entity to be created. Multiple entities cannot be created in a single request, only a single entity is supported.

The following fields should not be supplied in the create request as they are populated automatically by the system.

- entityId
- entityVersion
- lastModifiedDate
- lastModifiedUser
- createdDate
- createdUser

The response is named create<EntityType>Response and contains the entityId of the created entity if the operation was successful. The entityId returned is used in subsequent get and delete operations.

If the create operation fails, the response contains a Fault with a faultCode, faultString, and extra CrudFault details.

Example 26–1  Create Request

```
<v1:createDisTagRequest>
  <v1:disTag>
    <v13:name>Sample Tag</v13:name>
    <v13:description>Created through Web Service</v13:description>
  </v1:disTag>
</v1:createDisTagRequest>
```
Example 26–2  Create Response

Example 26–3  Create Failure (a name was not specified for the tag)

Entity Type Support
Supported on Entity Types

- DisBlackoutSchedule
- DisTag
- DisConfig

Get Operation

The get operation is used to retrieve an entity from the system. The get request requires a unique entity id and the entity details are returned in the response. For example, the getDisBlackoutSchedule operation returns all the details of a specific Blackout Schedule in the system.

The request for the Get operation is named get<EntityType>Request. The request contains a single entityId of the entity to be retrieved. Only one entityId can be specified in the request, multiples are ignored. The exception to this is the getRootEntity and getResultEntity operations; these operations accept multiple entityId id values.

If the entityId provided is not found in the system a fault is returned, not an empty response.

The response is named get<EntityType>Response and contains the details of the entity retrieved from the system.

If the get operation fails, the response contains a Fault with a faultCode, faultString, and extra CrudFault details.

Example 26–4  Get Request

Example 26–2  Create Response

Example 26–3  Create Failure (a name was not specified for the tag)
Get Operation

Example 26–5  Get Response

```xml
<ns118:getDisTagResponse>
  <ns118:disTag>
    <ns2:entityId>9586</ns2:entityId>
    <ns2:entityVersion>1</ns2:entityVersion>
    <ns12:parentRef>
      <ns2:entityId>9584</ns2:entityId>
    </ns12:parentRef>
    <ns12:name>Sample Child Tag</ns12:name>
    <ns12:description>Child Created through WS</ns12:description>
  </ns118:disTag>
</ns118:getDisTagResponse>
```

Example 26–6  Get Failure (entity id was not found)

```xml
<ns2:Fault>
  <faultcode>ns2:Server</faultcode>
  <faultstring>Cannot find Tag with entity Id 9586</faultstring>
  <detail>
    <ns158:crudFault>
      <ns151:rootStackTrace/>
    </ns158:crudFault>
  </detail>
</ns2:Fault>
```

Entity Type Support

Supported on Entity Types

- DisBlackoutSchedule
- DisTag
- DisConfig
- DisDiscrepancy
- DisInventoryImportPlugin
- DisNetworkDiscoveryPlugin
- DisAssimilationPlugin
- DisDiscrepancyResolutionPlugin
- DisDiscrepancyDetectionPlugin
- DisScanRun
- RootEntity
- ResultEntity
- Specification
- DefaultDisInventoryConfig
- DeviceInterface
- PhysicalDevice
- EquipmentHolder
- MediaInterface
Get All Operation

The get all operation is used to retrieve all entities of a certain type from the system. For example, the getAllDisBlackoutSchedule operation returns all the details of all the Blackout Schedules currently in the system.

This operation is only available for entities would not typically have many entries in the system and that do not support a Find operation.

The request for the Get All operation is named `getAll<EntityType>Request`. The request does not support any request parameters.

The response is named `getAll<EntityType>Response` and contains the details of all the entities retrieved from the system.

If the get operation fails, the response contains a Fault with a faultCode, faultString, and extra CrudFault details. Since this operation does not take any input parameters it should only fail due to environment or authentication issues.

**Example 26–7  Get All Request**

```xml
<v1:getAllRootDisTagsRequest/>
```

**Example 26–8  Get All Response**

```xml
<ns118:getAllRootDisTagsResponse>
    <ns118:rootDisTags>
        <ns2:entityId>9584</ns2:entityId>
        <ns2:entityVersion>3</ns2:entityVersion>
        ...etc
    </ns118:rootDisTags>
    <ns118:rootDisTags>
        <ns2:entityId>9585</ns2:entityId>
        <ns2:entityVersion>3</ns2:entityVersion>
        ...etc
    </ns118:rootDisTags>
</ns118:getAllRootDisTagsResponse>
```

Supported on Entity Types:

- DisBlackoutSchedule
- RootDisTag
- DisInventoryImportPlugin
- DisNetworkDiscoveryPlugin
- DisAssimilationPlugin
- DisDiscrepancyResolutionPlugin
- DisDiscrepancyDetectionPlugin
Delete Operation

The delete operation is used to remove an entity from the system. For example, the deleteDisBlackoutSchedule operation removes a particular Blackout Schedule from the system.

If successful, the result of the delete operation is immediately viewable in the Web UI.

The request for the Delete operation is named delete<EntityType>Request. The request contains a single entityId of the entity to be deleted. Only one entityId can be specified in the request, multiples are ignored.

If the entityId provided is not found in the system, or if the entity cannot be deleted, a fault is returned.

**Note:** The deleteDisConfig operation has an additional optional parameter you can enter in the delete request to force a scan to be deleted, even if it has associated discrepancies in the Running or Submitted state. See Table 26–1, "DisConfig Operations" for more information.

The response is named delete<EntityType>Response and contains the entityId of the entity deleted, which matches the id in the request.

If the delete operation fails, the response contains a Fault with a faultCode, faultString, and extra CrudFault details.

**Example 26–9 Delete Request**

```
<v1:deleteDisTagRequest>
  <v1:disTagRef>
    <v11:entityId>9579</v11:entityId>
  </v1:disTagRef>
</v1:deleteDisTagRequest>
```

**Example 26–10 Delete Response**

```
<ns118:deleteDisTagResponse>
  <ns118:disTagRef>
    <ns2:entityId>9579</ns2:entityId>
  </ns118:disTagRef>
</ns118:deleteDisTagResponse>
```

**Example 26–11 Delete Failure (entity id was not found)**

```
<ns2:Fault>
  <faultcode>ns2:Server</faultcode>
  <faultstring>Cannot find Tag with Entity Id9579</faultstring>
  <detail>
    <ns158:crudFault>
      <ns151:rootStackTrace/>
    </ns158:crudFault>
  </detail>
</ns2:Fault>
```

**Entity Type Support**

Supported on Entity Types:

- DisBlackoutSchedule
Update Operation

The update operation modifies an existing entity in the system. For example, the updateDisBlackoutSchedule operation updates a Blackout Schedule currently in the system.

If successful, the update is immediately available in the system and can be viewed in the Web UI.

The request for the Update operation is named update<EntityType>Request. The request must contain the full details of the new entity to be created, not just the fields that have changed. Multiple entities cannot be updated in a single request, only a single entity is supported. Unlike the create operation, the entityId must be supplied in the update operation to uniquely identify which entity to modify.

The entity version passed in the request must match the version that is held on the server. The entity version is incremented by the system every time the entity is modified. The entity version ensures that the entity has not been changed by some other user between when the entity was last retrieved and when updated. If the entity has been changed by some other user a Fault is returned as follows: Entity Version Mismatch: Input Version=1::Latest Version=2

Because the full details of the entity are required in the update request, the recommended steps are to do a get, get all, or find operation to get the details of the entity, and then copy these details into the update request, and modify the desired fields.

The following fields should not be supplied in the update request as they are populated automatically by the system or are not currently used.

- lastModifiedDate
- lastModifiedUser
- createdDate
- createdUser

The response is named update<EntityType>Response and contains the entityId of the updated entity if the operation was successful.

If the create operation fails, the response contains a Fault with a faultCode, faultString, and extra CrudFault details.

Example 26–12  Update Request

```xml
<v1:updateDisTagRequest>
  <v1:disTag>
    <v11:entityId>9586</v11:entityId>
    <v11:entityVersion>1</v11:entityVersion>
    <v12:parentRef>
      <v2:entityId>9584</v2:entityId>
    </v12:parentRef>
    <v11:name>Sample Child Tag</v11:name>
    <v11:description>Modified through WS</v11:description>
  </v1:disTag>
</v1:updateDisTagRequest>
```
Find Operation

Example 26–13  Update Response

<ns118:updateDisTagResponse>
  <ns118:disTagRef>
    <ns2:entityId>9586</ns2:entityId>
  </ns118:disTagRef>
</ns118:updateDisTagResponse>

Example 26–14  Update Failure (wrong entity version supplied)

<ns2:Fault>
  <faultcode>ns2:Server</faultcode>
  <detail>
    <ns158:crudFault>
      <ns151:rootStackTrace/>
    </ns158:crudFault>
  </detail>
</ns2:Fault>

Entity Type Support

Supported on Entity Types
- DisBlackoutSchedule
- DisTag
- DisConfig
- DisDiscrepancy

Find Operation

The find operation retrieves a list of entities that match filter search criteria. For example, the findDisConfig operation retrieves a list of DisConfig entities currently in the system that match a given set of search criteria.

The find operation in the Network Integrity API is equivalent in capability to the Search screens in the Network Integrity Web UI.

The request for the Find operation is named find<EntityType>Request. The find request can contain:
- From and To Ranges
- Sorting Fields (Ascending and Descending)
- Attribute Criteria
- Extended Attribute Criteria
- Criteria Operator (Equals, Contains, and so on)
- Conjunction Criteria (AND/OR)

Supported on Entity Types
- DisConfig
- DisScanRun
- DisDiscrepancy
**Find Operation**

**From and To Range**

The fromRange and toRange are used to limit the number of rows returned to a client. This is used to support paging in user interfaces using the Web Service API. It is also useful to improve performance and memory usage by retrieving many rows in smaller, more manageable chunks.

If the fromRange is not provided the default value is 0 which means the find returns the first row on. If the toRange is not provided in the request then the find operation is unbounded and returns all rows to the end.

```xml
<v1:findDisConfigRequest>
  <v1:disConfigSearchCriteria>
    <fromRange>0</fromRange>
    <toRange>20</toRange>
    <descending>name</descending>
    <disConfigConjunctionCriteriaItem>
      <nameAttributeCriteria>
        <value>Cisco</value>
        <operator>EQUALS</operator>
      </nameAttributeCriteria>
      <conjunction>AND</conjunction>
    </disConfigConjunctionCriteriaItem>
  </v1:disConfigSearchCriteria>
</v1:findDisConfigRequest>
```

**Ascending & Descending**

The ascending and descending fields control how the entity results are sorted in the response. The ascending and descending fields hold the name of the attribute to be sorted on. Multiple ascending and descending fields can be specified to add more than one level of sorting. If both an ascending and descending sort field are not provided in the request then the order of the entities returned is not sorted, and returned in the order they are persisted.

```xml
<v1:findDisConfigRequest>
  <v1:disConfigSearchCriteria>
    <fromRange>0</fromRange>
    <toRange>20</toRange>
    <descending>name</descending>
    <disConfigConjunctionCriteriaItem>
      <nameAttributeCriteria>
        <value>Cisco</value>
        <operator>EQUALS</operator>
      </nameAttributeCriteria>
      <conjunction>AND</conjunction>
    </disConfigConjunctionCriteriaItem>
  </v1:disConfigSearchCriteria>
</v1:findDisConfigRequest>
```

**Attribute Criteria**

The attribute criteria is used to specify the field and value to match when performing the find operation. In addition, an operator needs to be specified in the attribute criteria to determine how the match is done (for example, EQUALS, NOT_EQUALS, and so on).

Zero or more attribute criteria are contained within an entity’s ConjunctionCriteriaItem.
The `<EntityType>ConjunctionCriteriaItem` element defines a list of valid
`<attributeName>AttributeCriteria` child elements. For example, the
disConfigConjunctionCriteriaItem has an attributeCriteria for every attribute that is
searchable, namely the nameAttributeCriteria, descriptionAttributeCriteria,
enabledAttributeCriteria, and so on.

For each attribute criteria the value to match and the operator to use to perform the
match. The operators that are valid depend on the attribute type. For a list of valid
operators refer to the operator section below.

You can use wildcards in the value field for attributes that are text types. The
supported wildcard characters are "*", "%", and ".", "*", and "%" both represent a
match of zero or more characters. "_" represents a match of any single character.
Wildcard characters can be escaped with a backslash "\". To insert a backslash in the
query, insert two backslashes "\\".

```xml
<v1:findDisConfigRequest>
  <v1:disConfigSearchCriteria>
    <fromRange>0</fromRange>
    <toRange>20</toRange>
    <descending>name</descending>
    <disConfigConjunctionCriteriaItem>
      <nameAttributeCriteria>
        <value>Cisco</value>
        <operator>EQUALS</operator>
      </nameAttributeCriteria>
      <nameAttributeCriteria>
        <value>Juniper</value>
        <operator>EQUALS</operator>
      </nameAttributeCriteria>
      <conjunction>OR</conjunction>
    </disConfigConjunctionCriteriaItem>
  </v1:disConfigSearchCriteria>
</v1:findDisConfigRequest>
```

**Multiple Attribute Criteria**

Multiple criteria for the same attribute can be passed in a single find operation. In the
example below the find request is looking for Scans that start with the name Cisco or
Juniper. It is necessary to specify the ‘OR’ conjunction in this scenario or no rows is
returned.

```xml
<v1:findDisConfigRequest>
  <v1:disConfigSearchCriteria>
    <fromRange>0</fromRange>
    <toRange>20</toRange>
    <descending>name</descending>
    <disConfigConjunctionCriteriaItem>
      <nameAttributeCriteria>
        <value>Cisco*</value>
        <operator>EQUALS</operator>
      </nameAttributeCriteria>
      <nameAttributeCriteria>
        <value>Juniper*</value>
        <operator>EQUALS</operator>
      </nameAttributeCriteria>
      <conjunction>OR</conjunction>
    </disConfigConjunctionCriteriaItem>
  </v1:disConfigSearchCriteria>
</v1:findDisConfigRequest>
```
**Extended Attribute Criteria**

Extended Attribute Criteria allow the client application to find entities based on the attribute values on related entities. For example, to find all scans with a certain Scope Address would not be possible without extended criteria because the scope address is not defined on the DisConfig entity. Multiple criteria for the same attribute can be passed in a single find operation.

In the example below, the scope relationship on the DisConfig entity is followed, and then the addresses relationship if followed on the DisScope, to specify the addresses to match against. This search finds DisConfig entities that have either the address 10.156.68.136 or 10.156.68.140 in the scope. The schemas for the Web Service API define all the relationships and attributes that can be specified in the find operation.

```xml
<v1:findDisConfigRequest>
  <v1:disConfigSearchCriteria>
    <fromRange>0</fromRange>
    <toRange>20</toRange>
    <disConfigConjunctionCriteriaItem>
      <disConfigExtendedCriteriaItem>
        <scope>
          <disScopeConjunctionCriteriaItem>
            <disScopeExtendedCriteriaItem>
              <addresses>
                <disAddressConjunctionCriteriaItem>
                  <addressAttributeCriteria>
                    <value>10.156.68.136</value>
                    <operator>EQUALS</operator>
                  </addressAttributeCriteria>
                </disAddressConjunctionCriteriaItem>
                <disAddressConjunctionCriteriaItem>
                  <addressAttributeCriteria>
                    <value>10.156.68.140</value>
                    <operator>EQUALS</operator>
                  </addressAttributeCriteria>
                </disAddressConjunctionCriteriaItem>
              </addresses>
            </disScopeExtendedCriteriaItem>
          </disScopeConjunctionCriteriaItem>
        </scope>
      </disConfigExtendedCriteriaItem>
    </disConfigConjunctionCriteriaItem>
  </v1:disConfigSearchCriteria>
</v1:findDisConfigRequest>
```

**Criteria Operators**

The following are the allowed search operators for each entity and attribute. If the web service clients sends the wrong operator for a search criteria the web service search request fails and the client gets a message, which shows the allowed operators for that search criteria.

**DisConfig**

Table 26–10 shows the allowed search operators for DisConfig attributes.
Table 26–10  Allowed Search Operators for DisConfig Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>EQUALS</th>
<th>NOT_EQUAL</th>
<th>STARTS_WITH</th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Name</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanAction</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanType</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Source</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>NetworkAddress</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Enabled</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Run Reconciliation</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

DisScanRun

Table 26–11, Table 26–12, and Table 26–13 shows the allowed search operators for DisScanRun.

Table 26–11  Allowed Search Operators for DisScanRun Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>EQUALS</th>
<th>NOT_EQUAL</th>
<th>STARTS_WITH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td>Status</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanType</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td>Source</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanAction</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 26–12  Allowed Search Operators for DisScanRun Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>BEFORE</th>
<th>AFTER</th>
<th>ON_OR_AFTER</th>
<th>ON_OR_BEFORE</th>
<th>BETWEEN</th>
<th>NOT_BETWEEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScanStartTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ScanEndTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DiscrepancyDetectionStart Time</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DiscrepancyDetectionEndTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 26–13  Allowed Search Operators for DisScanRun Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>EQUALS</th>
<th>NOT_EQUAL</th>
<th>GREATER_THAN</th>
<th>LESS_THAN</th>
<th>BETWEEN</th>
<th>NOT_BETWEEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinorDiscrepancies</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MajorDiscrepancies</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CriticalDiscrepancies</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>WarningDiscrepancies</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
DisDiscrepancy

Table 26–14 and Table 26–15 shows the allowed search operators for DisDiscrepancy.

### Table 26–14  Allowed Search Operators for DisDiscrepancy Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>EQUALS</th>
<th>NOT_EQ</th>
<th>STARTS_WITH</th>
<th>IS_BLANK</th>
<th>IS_NOT_BLANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Severity</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Status</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ResolutionAction</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Owner</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Priority</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>EntityName</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanResultDetailName</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanType</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>EntityName</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanResultDetailName</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanName</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>EntityType</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CorrectedBy</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SubmittedBy</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ParentEntityName</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ParentEntityType</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Discovery/ImportValue</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Discovery/ImportSource</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanResultDetailCategory</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Type</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ScanType</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Table 26–15  Allowed Search Operators for DisDiscrepancy Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>BEFORE</th>
<th>AFTER</th>
<th>ON_OR_AFTER</th>
<th>ON_OR_BEFORE</th>
<th>BETWEEN</th>
<th>NOT_BETWEEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScanStartTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ScanEndTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DiscrepancyDetectionStart</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DiscrepancyDetectionEnd</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SubmittedTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LastStatusChangeTime</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
**Between/Not Between Operator**

When specifying the BETWEEN and NO_BETWEEN operators, two attribute criteria must be supplied or a fault is returned. The error message returned is **Incorrect number of values or incorrect format specified for attribute criteria: numberWarning.**

The following example searches for Scan Results that found between 10 and 100 discrepancy warnings.

```xml
<v1:findDisScanRunRequest>
  <v1:disScanRunSearchCriteria>
    <v11:fromRange>0</v11:fromRange>
    <v11:toRange>20</v11:toRange>
    <v11:disScanRunConjunctionCriteriaItem>
      <v12:disScanRunExtendedCriteriaItem>
        <v14:counts>
          <v119:disDiscrepancyCountsConjunctionCriteriaItem>
            <v120:warningAttributeCriteria>
              <v121:value>10</v121:value>
              <v121:value>100</v121:value>
              <v121:operator>BETWEEN</v121:operator>
            </v120:warningAttributeCriteria>
          </v119:disDiscrepancyCountsConjunctionCriteriaItem>
        </v14:counts>
      </v12:disScanRunExtendedCriteriaItem>
      <v12:conjunction>AND</v12:conjunction>
    </v11:disScanRunConjunctionCriteriaItem>
  </v1:disScanRunSearchCriteria>
</v1:findDisScanRunRequest>
```

**Date Criteria**

Date fields must be in the format mm/dd/yyyy mm:dd:ss AM/PM. The server time is always used for dates in Network Integrity. The following example searches for Scans Runs that started after the August 11th, 2010 10:00 am. Because the **AFTER** operator is used, scan runs that match this start time exactly are not included in the response. If operator **ON_OR_AFTER** was used then exact match start time scan runs would be included in the response.

```xml
<v1:findDisScanRunRequest>
  <v1:disScanRunSearchCriteria>
    <v11:fromRange>0</v11:fromRange>
    <v11:toRange>20</v11:toRange>
    <v11:disScanRunConjunctionCriteriaItem>
      <v12:discoveryBeginTimeAttributeCriteria>
        <v13:value>08/11/2010 10:00:00 AM</v13:value>
        <v13:operator>AFTER</v13:operator>
      </v12:discoveryBeginTimeAttributeCriteria>
    </v11:disScanRunConjunctionCriteriaItem>
    <v12:disScanRunSearchCriteria>
  </v1:disScanRunRequest>
```

**Conjunction Criteria**

The conjunction must be either AND or OR. Only the top level conjunction is used, conjunctions on lower level elements are ignored.
The conjunction appears at many levels in the find hierarchy. The conjunction at lower levels controls how the criteria at lower levels are evaluated logically.

In the following example the inner conjunction is OR because this request is designed to find any ScanRun that has discrepancy, regardless of severity. Notice the outer conjunction that has the value AND, this has no effect on the extended attribute criteria.

To change this find so it only finds scan runs that have a discrepancy of every severity, the inner conjunction on the disDiscrepancyCountsConjunctionCriteriaItem element would be changed to AND.

```xml
<v1:findDisScanRunRequest>
  <v1:disScanRunSearchCriteria>
    <v1:disScanRunConjunctionCriteriaItem>
      <v1:disScanRunExtendedCriteriaItem>
        <v1:counts>
          <v1:disDiscrepancyCountsConjunctionCriteriaItem>
            <v1:criticalAttributeCriteria>
              <v1:value>0</v1:value>
              <v1:operator>GREATER_THAN</v1:operator>
            </v1:criticalAttributeCriteria>
            <v1:minorAttributeCriteria>
              <v1:value>0</v1:value>
              <v1:operator>GREATER_THAN</v1:operator>
            </v1:minorAttributeCriteria>
            <v1:warningAttributeCriteria>
              <v1:value>0</v1:value>
              <v1:operator>GREATER_THAN</v1:operator>
            </v1:warningAttributeCriteria>
          </v1:disDiscrepancyCountsConjunctionCriteriaItem>
        </v1:counts>
      </v1:disScanRunConjunctionCriteriaItem>
    </v1:disScanRunSearchCriteria>
  </v1:disScanRunConjunctionCriteriaItem>
</v1:findDisScanRunRequest>
```
Find Response

The Find response contains all the details of the entities that matched the attribute criteria. The response only contains the number of entities defined by the from an to range. Subsequent find operations may be called to get all the entities depending on the number of rows matching the search criteria and the from and to range specified.

```xml
<ns118:findDisConfigResponse>
  <ns118:disConfigs>
    <ns2:entityId>9612</ns2:entityId>
    <ns2:entityVersion>1</ns2:entityVersion>
    <ns4:tagsRef>
      <ns2:entityId>9584</ns2:entityId>
    </ns4:tagsRef>
    <ns4:tagsRef>
      <ns2:entityId>9586</ns2:entityId>
    </ns4:tagsRef>
    <ns7:parameterGroups>
      <ns2:entityId>9606</ns2:entityId>
    </ns7:parameterGroups>
    <ns7:enabled>YES</ns7:enabled>
    <ns7:dataSource>TRUE</ns7:dataSource>
    <ns7:startScanReady>true</ns7:startScanReady>
  </ns118:disConfigs>
</ns118:findDisConfigResponse>
```

...and so on.

```xml
<ns7:enabled>YES</ns7:enabled>
<ns7:dataSource>TRUE</ns7:dataSource>
<ns7:startScanReady>true</ns7:startScanReady>
</ns118:disConfigs>
</ns118:findDisConfigResponse>
```

Special Function Operations

There are a few Web Service API operations that do not follow the standard pattern and are designed for a special purpose. The special function operations are:

- startScan
- stopScan
- getLatestScanStatus
- submitDisDiscrepancyResolutionOperations

These operations are described in the following sections.

Start Scan

The startScan operation starts a scan for a given DisConfig entityId. This operation is identical to the Start Scan operation in the Network Integrity Web UI. The request expects a DisConfig entityId and the response contains the entityId of the DisScanRun that was created for the scan.

**Example 26–15 Request:**

```xml
<v1:startScanRequest>
  <v1:disConfigRef>
    <v11:entityId>9612</v11:entityId>
  </v1:disConfigRef>
</v1:startScanRequest>
```

**Example 26–16 Response:**

```xml
<ns118:startScanResponse>
```

```xml
</ns118:startScanResponse>
```
Stop Scan

The stopScan operation stops a scan for a given DisConfig entityId. This operation is identical to the Stop Scan operation in the Network Integrity Web UI. The request expects a DisConfig entityId and the response contains the entityId of the DisScanRun that was created for the scan.

Example 26–17 Request:

```xml
<v1:stopScanRequest>
  <v1:disConfigRef>
    <v11:entityId>9612</v11:entityId>
  </v1:disConfigRef>
</v1:stopScanRequest>
```

Example 26–18 Response:

```xml
<ns118:stopScanResponse>
  <ns118:disScanRunRef>
    <ns2:entityId>13846</ns2:entityId>
  </ns118:disScanRunRef>
</ns118:stopScanResponse>
```

Get Latest Scan Status

The getLatestScanStatus returns the status of the latest run of a scan. The operation is equivalent to the information displayed in the Status section of the Manage Scans page of the Network Integrity UI. Information about the scan follows the status, including the number of discrepancies found, and the start time and duration of the scan.

This method is more efficient to call to monitor the running of a scan rather than call findDisScanRun many times.

Example 26–19 Request:

```xml
<v1:getLatestScanStatusRequest>
  <v1:disConfigRef>
    <v11:entityId>9612</v11:entityId>
  </v1:disConfigRef>
</v1:getLatestScanStatusRequest>
```

Example 26–20 Response (Running Scan):

```xml
<ns118:getLatestScanStatusResponse>
  <ns118:scanStatus>
    <ns120:discrepancySeverityCounts>
      <ns2:entityId>0</ns2:entityId>
      <ns2:entityVersion>0</ns2:entityVersion>
      <ns56:numberWarning>0</ns56:numberWarning>
      <ns56:numberMinor>0</ns56:numberMinor>
      <ns56:numberMajor>0</ns56:numberMajor>
      <ns56:numberCritical>0</ns56:numberCritical>
    </ns120:discrepancySeverityCounts>
    <ns120:discoveryWorkCounts>
      ...
    </ns120:discoveryWorkCounts>
  </ns118:scanStatus>
</ns118:getLatestScanStatusResponse>
```
<ns121:totalNoOfWorkItems>2</ns121:totalNoOfWorkItems>
<ns121:noOfCompletedWorkItems>0</ns121:noOfCompletedWorkItems>
<ns121:noOfFailedWorkItems>0</ns121:noOfFailedWorkItems>
<ns121:noOfInProgressWorkItems>2</ns121:noOfInProgressWorkItems>
<ns121:startTime>07/16/2010 11:17:05</ns121:startTime>
<ns121:duration/>
</ns120:discoveryWorkCounts>
<ns120:discrepancyWorkCounts>
<ns121:totalNoOfWorkItems>0</ns121:totalNoOfWorkItems>
<ns121:noOfCompletedWorkItems>0</ns121:noOfCompletedWorkItems>
<ns121:noOfFailedWorkItems>0</ns121:noOfFailedWorkItems>
<ns121:noOfInProgressWorkItems>0</ns121:noOfInProgressWorkItems>
<ns121:duration/>
</ns120:discrepancyWorkCounts>
<ns120:jobStateString>Running</ns120:jobStateString>
<ns120:discrepancyDetectionEnabled>true</ns120:discrepancyDetectionEnabled>
</ns118:scanStatus>
</ns118:getLatestScanStatusResponse>

Example 26–21 Response (Completed Scan)

<ns118:getLatestScanStatusResponse>
<ns118:scanStatus>
<ns120:discrepancySeverityCounts>
<ns2:entityId>15456</ns2:entityId>
<ns2:entityVersion>1</ns2:entityVersion>
<ns55:numberWarning>1</ns55:numberWarning>
<ns55:numberMinor>0</ns55:numberMinor>
<ns55:numberMajor>0</ns55:numberMajor>
<ns55:numberCritical>0</ns55:numberCritical>
</ns120:discrepancySeverityCounts>
<ns120:discoveryWorkCounts>
<ns121:totalNoOfWorkItems>2</ns121:totalNoOfWorkItems>
<ns121:noOfCompletedWorkItems>2</ns121:noOfCompletedWorkItems>
<ns121:noOfFailedWorkItems>0</ns121:noOfFailedWorkItems>
<ns121:noOfInProgressWorkItems>0</ns121:noOfInProgressWorkItems>
<ns121:startTime>07/16/2010 11:59:26</ns121:startTime>
<ns121:endTime>07/16/2010 11:59:52</ns121:endTime>
<ns121:duration>26s</ns121:duration>
</ns120:discoveryWorkCounts>
<ns120:discrepancyWorkCounts>
<ns121:totalNoOfWorkItems>2</ns121:totalNoOfWorkItems>
<ns121:noOfCompletedWorkItems>2</ns121:noOfCompletedWorkItems>
<ns121:noOfFailedWorkItems>0</ns121:noOfFailedWorkItems>
<ns121:noOfInProgressWorkItems>0</ns121:noOfInProgressWorkItems>
<ns121:startTime>07/16/2010 11:59:52</ns121:startTime>
<ns121:endTime>07/16/2010 11:59:55</ns121:endTime>
<ns121:duration>3s</ns121:duration>
</ns120:discrepancyWorkCounts>
<ns120:jobStateString>Completed</ns120:jobStateString>
<ns120:discrepancyDetectionEnabled>true</ns120:discrepancyDetectionEnabled>
</ns118:scanStatus>
</ns118:getLatestScanStatusResponse>

Submit Discrepancies For Resolution Processing

The submitDisDiscrepancyResolutionOperations operation takes a list of discrepancy entityId's and submits these discrepancies to be processed by a resolution action. This is the same as the Submit discrepancies operation in the Network Integrity Web UI.
The discrepancies submitted must have a discrepancy status of IDENTIFIED and have an Operation populated or else a fault is returned. The status and operation of the discrepancy can be updated using the updateDisDiscrepancy operation.

This operation is a two step operation in the Network Integrity Web UI to first add discrepancies to the queue, and then submit them. In the Web Service API this is a single operation.

If the operation is successful, the entityIds of the discrepancies submitted is returned in the response.

After submitting the discrepancies the status of the discrepancies is set to SUBMITTED.

**Example 26–22 Request**

```
<v1:submitDisDiscrepancyResolutionOperationsRequest>
  <!--1 or more discrepancies: -->
  <v1:disDiscrepancyRef>
    <v11:entityId>15448</v11:entityId>
  </v1:disDiscrepancyRef>
</v1:submitDisDiscrepancyResolutionOperationsRequest>
```

**Example 26–23 Response**

```
<ns118:submitDisDiscrepancyResolutionOperationsResponse>
  <ns118:disDiscrepancyRef>
    <ns2:entityId>15448</ns2:entityId>
  </ns118:disDiscrepancyRef>
</ns118:submitDisDiscrepancyResolutionOperationsResponse>
```

**Example 26–24 Failure (one or more discrepancies not in IDENTIFIED status)**

```
<ns2:Fault>
  <faultcode>ns2:Server</faultcode>
  <faultstring>DISCREPANCY_RESOLUTION_INVALID_STATUS</faultstring>
  <detail>
    <ns127:crudFault>
      <ns119:rootStackTrace/>
    </ns127:crudFault>
  </detail>
</ns2:Fault>
```

### High Level Scenarios

The following sections describe how to use the Web Service API in an end-to-end fashion.

### Creating a Scan

A scan is created using the createDisConfig operation, but there may be data and entities to be created or retrieved before calling the createDisConfig operation.

**Prerequisites:**

- A plugin entity id is required to create a scan. The list of discovery, import, and assimilation plugins that are deployed in the system can be determined by calling `getAllDisInventoryImportPlugin`, `getAllDisNetworkDiscoveryPlugin`, and `getAllDisAssimilationPlugin`. 
The plug-in entity may define one or more plug-in parameters (for example, SnmpParameters) that it expects to be passed. If it does then the plug-in returned in the previous step has one or more specificationsRef elements in the response. The expected plug-in parameters can be determined by calling getSpecification to determine the available plug-in parameters. Some plug-in parameters are optional and some are mandatory.

For more information about the parameters returned by getSpecification, refer to your plug-in or cartridge documentation.

If the Scan is to be tagged on creation then the Tag entity ids must be retrieved using one of getAllRootDisTags, getDisTag, createDisTag

If the Scan is to have Blackout schedules on creation then the Blackout entity ids must be retrieved using one of getAllDisBlackoutSchedule, getDisBlackoutSchedule, createDisBlackoutSchedule

The response from the createDisConfig operation, if successful, is an entity id for the scan. The entity id is used for deleting, retrieving, starting, and stopping the scan.

Starting, Stopping, and Monitoring a Scan

The Scan can be started using startScan operation and the DisConfig entity id that was returned when it was created. (It is also possible to do a findDisConfig operation to get the entity id).

The start scan operation returns the Scan run entity id that is used to get in the getDisScanRun operation to monitor the status and results of the scan.

It is also possible to monitor the scan progress using the DisConfig entity id and the getLatestScanStatus. This operation is more efficient and reports the current status of the scan along with other details.

An in-progress scan can be stopped using the stopScan operation and the DisConfig entity id. When the operation returns the scan is transitioned to STOPPING state, and asynchronously transitions to STOPPED when all scan processes have ended.

Retrieving Scan Results/Data

The starting point for retrieving scan results is the DisScanRun entity. The entity id of the DisScanRun is returned when the scan was started, or can be determined by performing the findDisScanRun operation.

If the scan successfully discovers data the DisScanRun has one or more resultGroups that contain one or more rootEntityRefsRef. These ids are used in the getRootEntity call to retrieve the root of the discovered data. The getRootEntity operation, unlike other get calls, accepts multiple entity ids for retrieving all root entities in a single call.

The getRootEntity operation does not retrieve the complete tree of results for performance reasons and to limit scope of entity traversal. The response from getRootEntity often contains references to other entities. These entities can be retrieved using the generic getResultEntity operation, or by type-specific get operations (getLogicalDevice, getEquipment, getPhysicalDevice, getLogicalDevice, getEquipmentHolder, and so on).

Most result data entities have specifications. To get details about the specification the entity is using, the getSpecification operation can be called using the specificationRef on the entity.
Working with Discrepancies

The starting point for working with discrepancies is the DisScanRun entity. The entity id of the DisScanRun is returned when the scan was started, or can be determined by performing the findDisScanRun operation.

The list of discrepancies created in discrepancy detection is in the DisScanRun entity as discrepanciesRef ids. The DisDiscrepancy entity can be retrieved using the getDisDiscrepancy operation passing the discrepanciesRef from the DisScanRun entity. The discrepancies can also be found using the findDisDiscrepancy operation with search criteria.

Several fields on the discrepancy, including the status, operation (Resolution Action), owner, priority, reasonForFailure, and notes can be updated using the updateDisDiscrepancy operation.

Discrepancies can be submitted for resolution by calling the submitDisDiscrepancyResolutionOperations operation. The operation takes a list of discrepancies to be submitted in the request. Discrepancies must be in the status of IDENTIFIED and have an operation populated to be submitted.

Network Integrity Web Service Samples

Network Integrity includes example requests and responses of calling the web service API. Find these examples in the Network Integrity Web Service Samples ZIP file.

Contents of the Network Integrity Web Service Samples ZIP File

Table 26–16 describes the directories, files, and file contents for the Network Integrity Web Service Samples ZIP file.

<table>
<thead>
<tr>
<th>Directory/File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build.xml</td>
<td>An example ANT build script that shows how to run the client with an SSL keystore as a VM argument.</td>
</tr>
<tr>
<td>WSDL-Documentation.html</td>
<td>Generated WSDL documentation that shows all the available operations. A short description of each operation is provided. Full WSDL source is included for reference.</td>
</tr>
<tr>
<td>IntegrityWebserviceSoapUIProject.xml</td>
<td>SoapUI Project File</td>
</tr>
<tr>
<td>integrity-schema\wsdl\NetworkIntegrityControlService.wsdl</td>
<td>Web Service Definition (WSDL)</td>
</tr>
<tr>
<td>integrity-schema\referenceSchema</td>
<td>Supporting XML Schema files</td>
</tr>
<tr>
<td>integrity-schema\schema</td>
<td>Supporting XML Schema files</td>
</tr>
<tr>
<td>integrity-ws-client.jar</td>
<td>Jar file containing Java Client type generated from the WSDL</td>
</tr>
<tr>
<td>jaxb-bindings.xml</td>
<td>JAXB Binding file to adjust generated package names when generating client classes from WSDL. These bindings are required if not using the provided integrity-ws-client.jar and generating client class files using a web service client generation tool.</td>
</tr>
<tr>
<td>src\oracle\integrity\ws\client\NetworkIntegrityControlService.java</td>
<td>This is a client side proxy class to get port types. This is the class where policy files and other authentication details are set.</td>
</tr>
<tr>
<td>src\oracle\integrity\ws\test\SampleNIHttpClient.java</td>
<td>An example client java class that makes a web service call.</td>
</tr>
</tbody>
</table>
Sample Java Client

Included in the Web Service Samples ZIP file is a sample java client. The sample java code is included in the src directory and contains:

- a sample client side proxy for getting a port type and setting the required policies and authentication.
- a client class that calls the getAllDisNetworkDiscoveryPlugin operation and prints the result to standard out.

To compile the sample JAVA code, the following JAR files are necessary:

- weblogic.jar: available in WL_Home/server/lib/
- wseeclient.jar: available in WL_Home/server/lib/
- jrf.jar: available in MW_Home/oracle_common/modules/oracle.jrf_11.1.1/
- integrity-ws-client.jar: included the Network Integrity Web Service Samples ZIP file.

---

Note: The required web service policy, Wssp1.2-2007-Https-UsernameToken-Plain.xml is included in the wseeclient.jar.

---

To run the sample JAVA code, you must run it with a full installation of WebLogic Server and ADF, because the JAR files referenced during compile require other JAR files. Set your classpath to point to the above JAR files in their installed location on your system. This can be done by installing WebLogic and ADF on your development system or run the client on your Network Integrity server.

If you plan on running a Web service client to communicate with a Network Integrity server that does not have a valid SSL certificate, you must download your server certificate and save it to a file to be used by your client. Then use the following VM argument when running your client. In this example, a file called jssecacerts has the SSL key that was downloaded.

-Djavax.net.ssl.trustStore=jssecacerts

Sample Soap UI Project

A SoapUI project is provided in the Cartridge Developer package to give examples of all the Web Service API calls and examples of the responses. The SoapUI project is used to test various Web Service API call scenarios.

To install the Soap UI, use the following procedure:

1. Download and Install SoapUI 3.5.1 (newer versions of SoapUI may work with the bundled project file, but it has not been tested)
2. Launch SoapUI and select the menu option File -> Import Project
3. Select the file IntegrityWebserviceSoapUIProject.xml and click Open

Also in the project is a NetworkIntegrityControlMockService that simulates the real Web Service API. For each operation there is one or more example responses provided in the mock service. The number of example requests in the binding does not always match the number of responses because the responses would be the same structure with a different id returned (for example, Create Blackout Response).
You can use the provided example requests or create new requests right-clicking the operation and selecting “New Request”. This creates a new request with all fields populated with a question mark. Many of the example requests in the project require modification to execute successfully because the entityIds in the example does not match other systems.

The NetworkIntegrityControlMockService is used to view examples of Web Service API responses for different scenarios. The mock service can also be started to respond to Web Service API calls with mock responses. Refer to SoapUI documentation for more details if this is desired.

**Submitting Request to the Server**

To submit a request to the server you must do the following:

1. Ensure the request is valid and all mandatory attributes are set.
2. Ensure the username and password are set in the request. See the next section on how to add the username and password to the request for how this is done.
3. Add a new endpoint by clicking on the drop down at the top of the request and select **add new endpoint**.
4. Add a new endpoint with the following format:

   https://<Managed_Server>:<Port>/NetworkIntegrityApp-NetworkIntegrityControlWebService-context-root/NetworkIntegrityControlServicePortType

5. Click **Play** to submit the request.

**Specifying User Name and Password in Request**

To add the user name and password to a request.

1. Click **Auto** tab at the bottom of the request.
2. Enter the Username and Password that has access to login to the Network Integrity Web UI.
3. Right click the request and select **Add WSS Username Token**.
4. Accept the default **PasswordText** and select **OK**.

The following structure is added to the request.

```xml
<wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
  <wsse:UsernameToken wsu:Id="UsernameToken-4" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <wsse:Username>niuser</wsse:Username>
    <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">niuser123</wsse:Password>
    <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">ZS2K4yCqOoQgSKL9DetBW==</wsse:Nonce>
    <wsu:Created>2010-09-13T01:21:17.578Z</wsu:Created>
  </wsse:UsernameToken>
</wsse:Security>
```

5. Delete the **Nonce** and **Created** elements in the above example (highlighted in bold) to reduce errors on future calls.
This chapter describes an Oracle Communications Network Integrity Event Notification, Scan Complete Notification, which allows external components to receive asynchronous event notification messages about the completion of scans.

You can develop a client to monitor event notifications, to and trigger follow-on actions.

About Clients for Monitoring Scan Run Complete Notification Messages

You can develop a message-driven bean (MDB) or Java messaging system (JMS) client that listens to the Network Integrity Event Notification JMS topic (oracle/communications/integrity/EventNotificationTopic) for scan-complete notification messages. For example, you can write post-processing logic that listens for messages that trigger other scans or send emails or SMS messages using the MDB/JMS client.

Develop the MDB/JMS client to listen to the Network Integrity application server for the JMS topic. The client must belong to the NetworkIntegrityRole group to access the JMS topic. See Network Integrity System Administrator’s Guide for more information on the NetworkIntegrityRole group.

Table 27–1 lists the properties used by EventNotificationTopic for client filtering.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Indicates the final scan run state:</td>
</tr>
<tr>
<td></td>
<td>■ COMPLETED</td>
</tr>
<tr>
<td></td>
<td>■ STOPPED</td>
</tr>
<tr>
<td></td>
<td>■ FAILED</td>
</tr>
<tr>
<td>Scan Action Name</td>
<td>Indicates the name of the scan action.</td>
</tr>
<tr>
<td>Scan Action Type</td>
<td>Indicates the type of the scan action:</td>
</tr>
<tr>
<td></td>
<td>■ NETWORK_DISCOVERY</td>
</tr>
<tr>
<td></td>
<td>■ INVENTORY_IMPORT</td>
</tr>
<tr>
<td></td>
<td>■ ASSIMILATION</td>
</tr>
</tbody>
</table>
Notification Messages also contain other properties which may be useful to you. For example, the ScanRunId can be obtained from the message body, which is used to retrieve additional information about the scan run.

The following example is a sample MDB/JMS client implementation model:

```java
package model;

import javax.annotation.Resource;
import javax.annotation.security.RunAs;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import weblogic.javaee.MessageDestinationConfiguration;

@MessageDriven(activationConfig =
    { @ActivationConfigProperty(propertyName = "connectionFactoryJndiName",
        propertyValue = "oracle/communications/integrity/NIXATCF"),
    @ActivationConfigProperty(propertyName = "destinationName", propertyValue =
        "oracle/communications/integrity/EventNotificationTopic"),
    @ActivationConfigProperty(propertyName = "destinationType",
        propertyValue = "javax.jms.Topic")
    }, mappedName = "oracle/communications/integrity/EventNotificationTopic")
@MessageDestinationConfiguration(connectionFactoryJNDIName =
    "oracle/communications/integrity/NIXATCF")
@RunAs("NetworkIntegrityRole")
public class MyEjbTestBean implements MessageListener {
    @Resource
    javax.ejb.MessageDrivenContext context;
    public void onMessage(javax.jms.Message message) {
        TextMessage text = (TextMessage)message;
        try {
            // write post-processing logic here
            // like trigger other scans, or send e-mails or SMS messages
            System.out.println("entered mdb.... ");
            System.out.println("received the following message: ");
            System.out.println("Status : "+text.getStringProperty("Status"));
            System.out.println("Scan_Action_Name : "+text.
                getStringProperty("Scan_Action_Name");
            System.out.println("Scan_Action_Type : "+text.
                getStringProperty("Scan_Action_Type");
            System.out.println("Discrepancy_Detection : "+text.
                getBooleanProperty("Discrepancy_Detection");
            System.out.println("scan txt : "+text.getText());
        }
```
} catch (JMSException e) {
    // Add log statements here
}
}
This chapter provides tips and tricks for working with processors, Design Studio, compiling, and building in Oracle Communications Network Integrity.

**Processors**

The following sections provide tips and tricks for working with processors:

- Creating Processors
- Specifying Context Parameters before Creating Implementation Class

**Creating Processors**

It is recommended to create a Processor from Action Editor (in the Processors tab). This way, the newly created Processor is automatically added to the Action. A processor not associated with any Action is an orphaned Processor. In general an orphaned processor results in a warning message, unless it is an orphaned SNMP Processor. In this case the processor results in an error message. Creating the Processor from Action Editor eliminates the possibility of orphaned Processors in a Network Integrity project.

**Specifying Context Parameters before Creating Implementation Class**

When creating a new processor, it is a good practice to properly configure the Context Parameters before saving the Processor. This way Design Studio properly generates the skeleton implementation Java class for the Processor with the correct input and output parameters. If the input and output context parameters are modified later, the generated Interface changes, but Design Studio does not automatically update the implementation class. The user must manually update the implementation class to comply with the changed interface.

**Working in Design Studio**

The following sections provides tips and tricks for working in Design Studio:

- About Package Explorer View and Navigator View
- About Error Log View
- Opening with the Wrong Editor
About Package Explorer View and Navigator View
These two views are useful for browsing all cartridge artifacts created by you or generated by Design Studio. There are also a few operations that are only accessible from these two views.

To open the Package Explorer View, select the menu item Window, then Show View, then Package Explorer.

To open the Navigator View, select the menu item Window, then Show View, then Other, then open General, and select Navigator.

About Error Log View
If there are problems with the project, but there is no Error Problem Marker, you may examine the Error Log View. This view reports problems and events from Eclipse plug-ins, including the Design Studio plug-ins.

To open the Error Log View, select the menu item Window then Show View then Other, then open General and select Error Log.

Error Log View shows events as they happen, and some of the displayed events may have occurred hour ago. It is usually a good idea to clear all logs first, and then do a Clean and Build to recreate a problem.

To clear the logs, click the Clear Log View icon on the tool bar. The icon is to the left of the Delete icon (red X).

Opening with the Wrong Editor
Sometimes, when you open an item in the cartridge project, the item is displayed using an editor that is not the default editor (for example, XML Editor instead of a Discovery Action Editor).

To reset the open behavior, go to the Package Explorer or the Navigator View. Then, open the tree for your project, and open the model folder. Find the item that you want to correct, and right-click it to bring up its context menu. From this menu, select Open With then Default Editor, and the open behavior is reset to default.

Compile or Build Problems
The following sections provides tips and tricks for working in Design Studio:

- About Java Errors in the Generated Controller Class
- Renaming or Deleting Actions and Processors
- Adding External Libraries to a Java Build Path
- About “Missing Required Library” Errors for External Libraries
- Error Marker on Cartridge but not on any Entities
- Unsealing a Sealed Cartridge
- Cartridge Failing to Build After Being Imported

About Java Errors in the Generated Controller Class
Compile errors in the generated Controller class of an Action usually mean that there are errors in the configuration of the Processor table of that Action. Look for a Studio
Error on an Action or Processor involved in the processor chain. Correct the error, then save all files and perform a clean operation to regenerate all generated files.

Renaming or Deleting Actions and Processors
When renaming an Action or a Processor, Design Studio only renames and refactors the generated Java source code. Likewise, when deleting an Action or a Processor, Design Studio only deletes the generated Java source code. These changes result in errors remaining in the processor implementation code and they must be corrected manually.

Adding External Libraries to a Java Build Path
To add an external library to the project for use by a processor, you must first copy the JAR file into the lib directory of the cartridge project. Then, you must add an entry for this library into the project’s Java Build Path. This can only be done in the Package Explorer or the Navigator View.

From either view, right-click the project and select Properties. In the Properties dialog, select Java Build Path in the left side, and select the Libraries tab. Now you can select Add External Jars to add your libraries.

About “Missing Required Library” Errors for External Libraries
You have copied the required library JAR files into the lib directory of your cartridge project, and you have added these libraries into your project’s Java Build Path. If you are still getting missing required library errors, refresh your cartridge project to cause Design Studio to notice the added library.

To refresh your project, go to the menu Windows then Show View then open Package Explorer then right-click your project, and select Refresh. Follow this with a Clean and Build of the project.

Error Marker on Cartridge but not on any Entities
If there is an error marker on the cartridge itself, but there are no error marker on any cartridge entities (Actions, Processors, Model Collections, and so on), then try checking the cartridge project using the Package Explorer View or the Navigator View. Sometimes the error markers are on some generated artifacts instead.

If there are no error markers on anything else, then try a Refresh and Rebuild operation. Go into Package Explorer or Navigator view, right-click the top-level project, and select Refresh. Then, choose the menu Project, then Clean, and choose to clean and rebuild all projects.

Unsealing a Sealed Cartridge
You may encounter build problems if you unseal a sealed cartridge in your workspace. The error logs may indicate that some dependent JAR files are missing from the workspace. The main cause for this is that the sealed cartridge may not have included any source code, and that a Clean operation may delete the JAR file, and then is not able to recreate it. The solution is to delete the unsealed cartridge, and re-import the sealed cartridge.
Cartridge Failing to Build After Being Imported

In some cases, Design Studio may not be able to build a cartridge successfully immediately after it is imported. The Problem View shows problem markers against Design Studio entities or JAVA classes.

To successfully build the cartridge:

1. Ensure all design time cartridge dependencies are satisfied and that all dependent cartridges exist in the workspace and are building successfully.

2. From the **Project** menu, enable **Build Automatically**.

3. From the **Project** menu, select **Clean** to clean and rebuild the cartridge.

4. Verify that the problem markers no longer appear in the Problem View. If the problems still appear:
   a. Close the cartridge project.
   b. Re-open the cartridge project.
   c. Clean the project.

5. Verify that the problem markers no longer appear in the Problem View. If the problems still appear:
   b. Re-open Design Studio.
   c. Clean the project.

6. Verify that the problem markers no longer appear in the Problem View. If the problems still appear:
   a. Back up any changes to the cartridge.
   b. Delete the cartridge project.
   c. Re-import the cartridge project.
This chapter provides information about the Oracle Communications Network Integrity plug-in validation error messages.

Error Message Classifications and Conditions

Table 29–1 lists the error messages, error classifications, and error conditions for the Network Integrity plug-in.

Note: Text inside {} represents a variable that is replaced based on the current error condition.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Classification</th>
<th>Error Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action names must start with a letter.</td>
<td>Error</td>
<td>Occurs when creating an Action without a letter as the first character in the Name.</td>
</tr>
<tr>
<td>The character {character} is not valid in an implementation prefix.</td>
<td>Error</td>
<td>Occurs when the implementation prefix of an Action or Processor contains characters that cannot be part of a Java identifier.</td>
</tr>
<tr>
<td>Processor {processor name} already has more than one parent Action assigned.</td>
<td>Error</td>
<td>Occurs if an attempt is made to associate a Processor to a second Action. A Processor can have only one parent.</td>
</tr>
<tr>
<td>Processor Parameter: {parameter name} not found in Parameter list for Processor {processor name}.</td>
<td>Informational</td>
<td>Occurs if an attempt is made to rename a Processor’s Input or Output Parameter, which no longer exists in the Parameter list.</td>
</tr>
<tr>
<td>Processor Property Group: {property group name} not found in Property Group list for Processor {processor name}.</td>
<td>Informational</td>
<td>Occurs if an attempt is made to rename a Processor’s Property Group, which no longer exists in the Property Group list.</td>
</tr>
<tr>
<td>Action Condition {condition name} not found in Condition list for Action {action name}.</td>
<td>Informational</td>
<td>Occurs if an attempt is made to rename an Action’s Condition, which no longer exists in the Condition list.</td>
</tr>
<tr>
<td>The generated implementation prefix for this entity conflicts with the implementation prefix of entity &quot;{entity name}&quot;. Choose a different name.</td>
<td>Error</td>
<td>Occurs when an Action’s or Processor’s Implementation Prefix conflicts with an existing prefix.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Classification</td>
<td>Error Condition</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cannot get cartridge from Action: <em>(action name)</em>.</td>
<td>Error</td>
<td>Occurs when Studio is unable to determine the Cartridge to which the current Action belongs as part of dependency checks before building.</td>
</tr>
<tr>
<td>SNMP Parameters cannot be added to the Discovery Action because the project does not have a data dictionary.</td>
<td>Warning</td>
<td>Occurs when a SNMP Processor is created and no Data Dictionary exists with the Integrity Project. To correct, first create a Data Dictionary, then create the SNMP Processor.</td>
</tr>
<tr>
<td>SNMP Parameters cannot be added to the Discovery Action because a DataDictionary Element matching the name SnmpParameters was found but it is not assigned the UI Parameter Specification type.</td>
<td>Warning</td>
<td>Occurs when a SNMP Processor is created and the Integrity Project’s Data Dictionary exists with an SnmpParameters structure that is not of Entity Type UI Parameter Specification. To correct, delete the conflicting SnmpParameters or change its Entity Type to UI Parameters Specification.</td>
</tr>
<tr>
<td>SNMP Processor has not specified any OIDs</td>
<td>Error</td>
<td>Occurs if an SNMP Processor has not specified any OIDs.</td>
</tr>
<tr>
<td>Processor implementation has not been specified</td>
<td>Error</td>
<td>Occurs if the Processor’s Implementation Class is not specified on the Processor’s Details tab.</td>
</tr>
<tr>
<td>Processor implementation is missing</td>
<td>Error</td>
<td>Occurs if the Processors implementation class, which is specified on the Processor’s Details tab, is missing in Studio.</td>
</tr>
<tr>
<td>Processor implementation package does not match Processor interface package</td>
<td>Error</td>
<td>Occurs if the package defined in the Processor’s Implementation Class does not match the package of the Processor’s generated interface.</td>
</tr>
<tr>
<td>MIB Directory has not been specified. See Oracle Design Studio Network Integrity preferences.</td>
<td>Error</td>
<td>Occurs if the MIB Directory is not specified in the Oracle Design Studio Network Integrity Preferences (Window -&gt; Preferences -&gt; Oracle Design Studio -&gt; Network Integrity).</td>
</tr>
<tr>
<td>MIB directory &quot;<em>[mib directory]</em>&quot; does not exist. See Oracle Design Studio Network Integrity preferences</td>
<td>Error</td>
<td>Occurs if the MIB Directory as specified in the Oracle Design Studio Network Integrity Preferences (Window -&gt; Preferences -&gt; Oracle Design Studio -&gt; Network Integrity) does not exist.</td>
</tr>
<tr>
<td>MIB module &quot;<em>[mib module name]</em>&quot; does not exist</td>
<td>Error</td>
<td>Occurs if the MIBs specified as part of the SNMP Processor are not available in the MIB Directory.</td>
</tr>
<tr>
<td>Processor is not used in an Action</td>
<td>Warning</td>
<td>Occurs if the Processor is not used by an Action.</td>
</tr>
<tr>
<td>Action has not specified a result category. At least one result category must be specified</td>
<td>Error</td>
<td>Occurs if the Action has not defined at least one Result Category.</td>
</tr>
<tr>
<td>Action has not specified a result source. At least one result source must be specified</td>
<td>Error</td>
<td>Occurs if the Discrepancy Detection Action does not contain at least one Result Source.</td>
</tr>
<tr>
<td>Result source Action &quot;<em>[action name]</em>&quot; cannot be found</td>
<td>Error</td>
<td>Occurs if the Discrepancy Detection Action’s Result Source Action cannot be found. For example, the Action has been deleted.</td>
</tr>
<tr>
<td>Result source &quot;<em>[action name] [result source name]</em>&quot; cannot be found</td>
<td>Error</td>
<td>Occurs if the Discrepancy Detection Action’s Result Source cannot be found. For example, it has been deleted from the Action.</td>
</tr>
</tbody>
</table>
### Table 29–1 (Cont.) Network Integrity Error Message, Classification, and Error Condition

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Classification</th>
<th>Error Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI parameter &quot;{parameter name}&quot; does not exist</td>
<td>Error</td>
<td>Occurs if the Data Dictionary Structure referenced by an Action’s UI Parameters has been deleted.</td>
</tr>
<tr>
<td>Data dictionary element for UI parameter &quot;{parameter name}&quot; is invalid</td>
<td>Error</td>
<td>Occurs if the Data Dictionary Structure or its Elements are invalid. For example, the Entity Type is not UI Parameter Specification.</td>
</tr>
<tr>
<td>SNMP Processor requires “SnmpParameters” UI parameter</td>
<td>Error</td>
<td>Occurs if the SnmpParameters UI Parameters are not available in the workspace. To correct, ensure the MIB_IL_SNMP_Cartridge is imported in the workspace. Next, remove and re-add the SNMP Processor to the Discovery Action.</td>
</tr>
<tr>
<td>Address handler implementation has not been specified</td>
<td>Error</td>
<td>Occurs if the Implementation Class for an AddressHandler is not specified.</td>
</tr>
<tr>
<td>Address handler implementation is missing</td>
<td>Error</td>
<td>Occurs if the Implementation class itself is not in Studio.</td>
</tr>
<tr>
<td>Address handler implementation package does not match interface package</td>
<td>Error</td>
<td>Occurs if the package defined in the AddressHandler’s Implementation Class does not match the package of the AddressHandler’s generated interface.</td>
</tr>
<tr>
<td>Specification &quot;{specification name}&quot; does not exist</td>
<td>Error</td>
<td>Occurs if the Specification referenced by a Processor’s Model Collection does not exist. For example, it has been deleted.</td>
</tr>
<tr>
<td>Data dictionary element for specification &quot;{specification name}&quot; is invalid</td>
<td>Error</td>
<td>Occurs if the Data Dictionary Element is invalid. For example, POMS does not support it.</td>
</tr>
<tr>
<td>Stale imported Action &quot;{action name}&quot;. The imported Action’s Processors have changed since they were imported.</td>
<td>Error</td>
<td>Occurs when imported Action’s Processors have changed. For example, the ordering of the Processors in the owning Action has changed.</td>
</tr>
<tr>
<td>Action contains no Processors</td>
<td>Warning</td>
<td>Occurs when an Action exists without any Processors.</td>
</tr>
<tr>
<td>Cartridge contains neither Actions nor address handlers</td>
<td>Error</td>
<td>Occurs when a new Integrity Project contains no Actions or AddressHandlers.</td>
</tr>
<tr>
<td>Provider has not been specified</td>
<td>Warning</td>
<td>Occurs when the Cartridge Provider has not be specified on the Network Integrity Cartridge Properties tab.</td>
</tr>
<tr>
<td>Cartridge cannot contain both Actions and Address Handlers</td>
<td>Error</td>
<td>Occurs when an Integrity Project contains both Address Handlers and Actions, which is invalid.</td>
</tr>
<tr>
<td>Condition implementation has not been specified for Condition &quot;{condition name}&quot;</td>
<td>Error</td>
<td>Occurs when the Implementation Class has not been provided for a Condition within an Action.</td>
</tr>
<tr>
<td>Condition implementation is missing for Condition &quot;{condition name}&quot;</td>
<td>Error</td>
<td>Occurs if the Implementation class itself is not in Studio.</td>
</tr>
<tr>
<td>Model Collection is not associated with any Actions. A model collection must be associated with at least one Action.</td>
<td>Error</td>
<td>Occurs when the Model Collection is not associated to at least one Action.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Classification</td>
<td>Error Condition</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Resolution Action has not specified a Resolution Action Label.</td>
<td>Error</td>
<td>Occurs when the Resolution Action does not have a Resolution Action Label, which is used as the resolution string in the UI for resolving discrepancies.</td>
</tr>
<tr>
<td>Error Retrieving Cartridge Model</td>
<td>Error</td>
<td>Occurs when a given Action’s Processors do not have a Provider.</td>
</tr>
<tr>
<td>Action &quot;{action name}&quot; is not a valid Action and cannot be added.</td>
<td>Error</td>
<td>Occurs when selecting an invalid Action when adding Processors to an Action.</td>
</tr>
<tr>
<td>Action &quot;{action name}&quot; does not contain any Processors. Actions must contain at least one processor to be eligible for inclusion in another Action.</td>
<td>Error</td>
<td>Occurs when importing an Action, which contains no Processors.</td>
</tr>
<tr>
<td>The are no output parameters on any of the Processors that are of a type that can be iterated over.</td>
<td>Error</td>
<td>Occurs when adding a For Each to an Action, which has Processors that do not have an output parameter that allows iteration.</td>
</tr>
<tr>
<td>The order of Processors from Imported Actions can not be changed.</td>
<td>Error</td>
<td>Occurs when the order of Processors from Imported Actions is changed.</td>
</tr>
<tr>
<td>Processor &quot;{processor name}&quot; uses parameter &quot;{parameter name}&quot; and Processor &quot;{processor name}&quot; outputs this parameter, continuing may make the Action invalid. Do you want to continue?</td>
<td>Confirmation</td>
<td>Occurs when changing the order (Moving Down) of Processors within an Action resulting in invalidating the flow of parameters thus making the Action as a whole invalid.</td>
</tr>
<tr>
<td>Processor &quot;{processor name}&quot; has a condition that uses parameter &quot;{parameter name}&quot; and Processor &quot;{processor name}&quot; outputs this parameter, continuing may make the Action invalid. Do you want to continue?</td>
<td>Confirmation</td>
<td>Occurs when changing the order of Processors (Moving Down) within an Action resulting in invalidating one or more conditions.</td>
</tr>
<tr>
<td>Processor &quot;{processor name}&quot; outputs parameter &quot;{parameter name}&quot; and Processor &quot;{processor name}&quot; uses this parameter, continuing may make the Action invalid. Do you want to continue?</td>
<td>Confirmation</td>
<td>Occurs when changing the order (Moving Up) of Processors within an Action resulting in invalidating the flow of parameters thus making the Action as a whole invalid.</td>
</tr>
<tr>
<td>Processor &quot;{processor name}&quot; outputs parameter &quot;{parameter name}&quot; and Processor &quot;{processor name}&quot; has a condition that uses this parameter, continuing may make the Action invalid. Do you want to continue?</td>
<td>Confirmation</td>
<td>Occurs when changing the order of Processors (Moving Up) within an Action resulting in invalidating one or more conditions.</td>
</tr>
<tr>
<td>Action should not be null</td>
<td>Error</td>
<td>Occurs when adding or removing elements (Processors, For Each blocks, and so on) from an Action, which is null.</td>
</tr>
<tr>
<td>The condition could not be added because the following {action name} are read only</td>
<td>Error</td>
<td>Occurs when attempting to add a Condition to an Action, which is read only.</td>
</tr>
<tr>
<td>The condition could not be removed because the following {action name} are read only</td>
<td>Error</td>
<td>Occurs when attempting to remove a Condition from an Action, which is read only.</td>
</tr>
<tr>
<td>The condition interface {condition interface name} has not been generated. It is recommended to save and build the Action before creating the implementation so that the interface is generated. Continue creating the implementation class anyway?</td>
<td>Confirmation</td>
<td>Occurs if the Condition’s interface has not been generated before the implementation class being generated.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Classification</td>
<td>Error Condition</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Condition '{condition name}' has relations. Are you sure you want to delete it?</td>
<td>Warning</td>
<td>Occurs when the Condition to be deleted has relationship to a Processor.</td>
</tr>
<tr>
<td>A Condition called {condition name} already exists on this plug-in, specify a different name.</td>
<td>Error</td>
<td>Occurs when attempting to create a Condition with a name that already exists within the Action.</td>
</tr>
<tr>
<td>The Condition Name must have a length greater than 0 but not exceeding 50 characters</td>
<td>Error</td>
<td>Occurs when the length of the target Condition name is not within the valid range of 1 – 50 characters.</td>
</tr>
<tr>
<td>This Output Parameter type is used by a for each, therefore the parameter type must be an iterable type.</td>
<td>Error</td>
<td>Occurs when the Output Parameter type used as an input to a For Each is not iterable.</td>
</tr>
<tr>
<td>Processor '{processor name}' is not writable, so references to this output parameter is not updated.</td>
<td>Error</td>
<td>Occurs when trying to modify a Processor, which is read only.</td>
</tr>
<tr>
<td>Input Parameters are referencing this Output Parameter. Changing the name or type may generate compile errors. Do you want to continue?</td>
<td>Warning</td>
<td>Occurs when changing the name of an Output Parameter, which has referencing Input Parameters on Processors whose java classes are already generated.</td>
</tr>
<tr>
<td>Input Parameters are referencing this Output Parameter. Removing it generates compile and validation errors. Do you want to continue?</td>
<td>Warning</td>
<td>Occurs when removing an Output Parameter, which has referencing Input Parameters on Processors whose java classes are already generated.</td>
</tr>
<tr>
<td>There are no output parameters available from preceding Processors to be selected</td>
<td>Informational</td>
<td>Occurs when selecting a Processor’s Input Parameters and no preceding Processor has an Output Parameters.</td>
</tr>
<tr>
<td>No uses of output parameter '{parameter name}' were found.</td>
<td>Informational</td>
<td>Occurs when viewing the Usage of an Output Parameter, which is not used as an Input Parameter.</td>
</tr>
<tr>
<td>The provided name already exists. Enter a different name.</td>
<td>Error</td>
<td>Occurs when adding a Condition using a name that already exists.</td>
</tr>
<tr>
<td>The Name cannot exceed 50 characters</td>
<td>Error</td>
<td>Occurs when adding a Condition with a name that exceeds 50 characters.</td>
</tr>
<tr>
<td>The name must start with a letter.</td>
<td>Error</td>
<td>Occurs when creating an Element (for example, Processor, AddressHandler) with an invalid name (i.e. starts with a digit) using the Studio Model Entity Wizard.</td>
</tr>
<tr>
<td>Action names must start with a letter.</td>
<td>Error</td>
<td>Occurs when creating an Action with an invalid name.</td>
</tr>
<tr>
<td>A value for implementation prefix is required when the use default option is not selected.</td>
<td>Error</td>
<td>Occurs when creating an Action and no implementation prefix is specified when the default option is not selected.</td>
</tr>
<tr>
<td>The implementation prefix must begin with a letter.</td>
<td>Error</td>
<td>Occurs when specifying an Action’s or Processor’s Implementation Prefix starting with a character other than a letter.</td>
</tr>
<tr>
<td>Error trying to lookup interface in project.</td>
<td>Error</td>
<td>Occurs when Studio is attempting to create a class that implements an interface, which does not exist in the Project.</td>
</tr>
<tr>
<td>An error occurred attempting to create a Java class. Details: {error details}</td>
<td>Error</td>
<td>Occurs when Studio is unable to create a Java class likely due to a Java Model problem or permissions.</td>
</tr>
</tbody>
</table>
Table 29–1 (Cont.) Network Integrity Error Message, Classification, and Error Condition

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Classification</th>
<th>Error Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The generated interface <em>(interface name)</em> could not be found in your project. It is recommended to save and build before creating the implementation so that the interface is available. Continue creating the implementation class anyway?</td>
<td>Warning</td>
<td>Occurs when generating the implementation before the interface is available. For example, when creating a new Processor, it is recommended to Save and Build before creating the implementation class.</td>
</tr>
<tr>
<td>The required interface, <em>(interface name)</em>, could not be found. Please clean and build the project.</td>
<td>Error</td>
<td>Occurs when selecting the implementation before the interface is available. For example, when creating a new Processor, it is recommended to Save and Build before selecting the implementation class.</td>
</tr>
<tr>
<td>The package rename cannot be performed because the following entities are not writable:</td>
<td>Error</td>
<td>Occurs when modified the Default Package property on the Network Integrity Cartridge Properties tab and the underlying classes are read only.</td>
</tr>
<tr>
<td>Project name should not contain spaces.</td>
<td>Error</td>
<td>Occurs when attempting to create a Integrity Project with a name that contains spaces.</td>
</tr>
<tr>
<td>A Default Cartridge Package is required</td>
<td>Error</td>
<td>Occurs if there is no Default Cartridge Package specified under the Oracle Design Studio -&gt; Network Integrity section in the Design Studio Preferences located under Window -&gt; Preferences.</td>
</tr>
<tr>
<td>Spaces are not allowed in the package name</td>
<td>Error</td>
<td>Occurs if the Default Cartridge Package value contains spaces.</td>
</tr>
<tr>
<td>This removes all generated UI hints artifacts. Do you wish to continue?</td>
<td>Confirmation</td>
<td>Occurs when clicking the Clean UI Hints button located on the UI Hints tab of the Network Integrity Cartridge element.</td>
</tr>
<tr>
<td>The UI Hints could not be cleaned, please ensure the mds.mar file is not read only</td>
<td>Error</td>
<td>Occurs when attempting to clean the UI Hints while the mds.mar file is read only. The mds.mar is located in the cartridge lib directory.</td>
</tr>
<tr>
<td>Spaces are not allowed in the package name</td>
<td>Error</td>
<td>Occurs when attempting to rename the Default Package property on the Properties tab of the Network Integrity Cartridge element.</td>
</tr>
<tr>
<td>Please fix fields with errors.</td>
<td>Error</td>
<td>Occurs when creating an Output Parameter with an invalid Type.</td>
</tr>
<tr>
<td>The first character in a parameter name should be lowercase</td>
<td>Warning</td>
<td>Occurs when adding Output Parameters to a Processor and the Parameter Name begins with an invalid character (i.e. uppercase).</td>
</tr>
<tr>
<td>A <em>[field name]</em> value must be entered.</td>
<td>Error</td>
<td>Occurs when adding Output Parameters, Property Groups and Properties to a Processor and no Name value is specified.</td>
</tr>
<tr>
<td>Parameter <em>[parameter name]</em> could not be added because a parameter with the same name already exists. Remove the parameter with the same name and retry the operation.</td>
<td>Error</td>
<td>Occurs when adding an Output or Input Parameter using a name that already exists in the Parameter list. Names must be unique in the parameter list since the Name is used to generate the getter methods.</td>
</tr>
<tr>
<td>The name &quot;<em>[parameter name]</em>&quot; already exists as a parameter, enter a different name</td>
<td>Error</td>
<td>Occurs when adding an Output Parameter using a name that already exists.</td>
</tr>
<tr>
<td>The name cannot contain spaces</td>
<td>Error</td>
<td>Occurs when adding an Output Parameter or Property Group to a Processor and the name contains spaces.</td>
</tr>
</tbody>
</table>
Error Message Classifications and Conditions

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Classification</th>
<th>Error Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name cannot start with a number</td>
<td>Error</td>
<td>Occurs when adding an Output Parameter or Property Group to a Processor and the name starts with a number.</td>
</tr>
<tr>
<td>Parameter Type &quot;{parameter type}&quot; may produce warnings in generated code.</td>
<td>Confirmation</td>
<td>Occurs if the Parameter Type of an Output or Input Parameter may cause compile warnings.</td>
</tr>
<tr>
<td>The Parameter Type &quot;{parameter type}&quot; produces the following warning in</td>
<td>Confirmation</td>
<td>Occurs if the generated code contains warnings based on the Parameter Type of an Output or Input Parameter.</td>
</tr>
<tr>
<td>generated code. Do you want to continue?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The name must be a valid java identifier that does not contain special</td>
<td>Error</td>
<td>Occurs when adding an Output Parameter or Property Group to a Processor and the name contains a special character (for example, %).</td>
</tr>
<tr>
<td>characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The name &quot;{parameter name}&quot; is a reserved word in Java, enter a different name</td>
<td>Error</td>
<td>Occurs when adding an Output Parameter or Property Group to a Processor and the name is equivalent to a reserved word in Java and therefore would cause compiling errors.</td>
</tr>
<tr>
<td>Type {parameter type} could not be found in the project</td>
<td>Error</td>
<td>Occurs if the Parameter Type of an Output or Input Parameter could not be found in the Integrity Project.</td>
</tr>
<tr>
<td>A property group with the name &quot;{property group name}&quot; already exists on</td>
<td>Error</td>
<td>Occurs when adding a Property Group using a name that already exists.</td>
</tr>
<tr>
<td>this input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A Property with the name &quot;{property name}&quot; and value &quot;{property value}&quot;</td>
<td>Error</td>
<td>Occurs when adding or modifying a Property using a name and value that already exists.</td>
</tr>
<tr>
<td>already exists, please choose a different name/value combination</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One or more errors exist with the fields</td>
<td>Error</td>
<td>Occurs when creating a Property Group with an invalid Name.</td>
</tr>
<tr>
<td>A Property Group with the name &quot;{property group name}&quot; already exists,</td>
<td>Error</td>
<td>Occurs when modifying a Property Group changing its name to a name that already exists.</td>
</tr>
<tr>
<td>please choose a different name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One or more errors exist with the fields</td>
<td>Error</td>
<td>Indicates a problem with Result Groups or Result Source.</td>
</tr>
<tr>
<td>A {field name} value must be entered.</td>
<td>Error</td>
<td>Occurs when creating a Result Category or Condition with no name.</td>
</tr>
<tr>
<td>The Result Category Name must have a length greater than 0 but not exceeding</td>
<td>Error</td>
<td>Occurs when modifying a Result Category changing its name to have a length of 0 or greater than 255 characters.</td>
</tr>
<tr>
<td>255 characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Dictionary named &quot;{data dictionary name}&quot; could not be found.</td>
<td>Error</td>
<td>Occurs when the Data Dictionary elements of a Model Collection cannot be found.</td>
</tr>
<tr>
<td>The MIB File &quot;{mib filename}&quot; could not be loaded because of the</td>
<td>Error</td>
<td>Occurs when a file other than a MIB File is selected when clicking the Load MIB button within an SNMP Processor.</td>
</tr>
</tbody>
</table>
| following error: 
| error details                                                                  |                |                                                                                                                                                 |
| A valid MIB Module called "{mib module name}" could not be found in MIB   | Error          | Occurs when the target MIB File attempting to be loaded by a SNMP Processor does not contain any MIB Modules.                                  |
| File: "{mib filename}"                                                      |                |                                                                                                                                                 |
When developing cartridges within Design Studio for Network Integrity it is likely that the developer requires logging for traceability during normal cartridge operation and for debugging. This section outlines how to introduce logging into the developer’s implementation. This section addresses logging that is visible inside the WebLogic log files. It does not discuss introducing Design Studio logging (for example, Design Studio Error Logs).

Network Integrity uses the java.util.logging package for logging messages. For an overview of the Java logging framework, visit Oracle’s site on the subject at http://download.oracle.com/javase/6/docs/api/index.html.

To create an instance of the appropriate logger add a static variable to an implementation class passing in the name of the current class. For example,

```java
private static final Logger logger = Logger.getLogger(DiscrepancyDetectorImpl.class.getName());
```

When the above is defined, invoke logging according to the API specification. For example,

```java
logger.log(Level.SEVERE, "Error while detecting discrepancies.", e);
```

To redirect the Network Integrity logs produced by the above into a WebLogic log file use the following procedure:

1. Insert the following 2 XML fragments into the file `<DOMAIN_HOME>/config/fmwconfig/servers/<TargetServer>/logging.xml`. The `<!--TargetServer-->` represents the name of the WebLogic Server where the Network Integrity application is running.
   a. The following fragment goes inside the `<log_handlers>` block and defines the log handler and log file location. If required, change the log handler; however, this value must match the value referenced in the fragment in step b. If necessary, change the location where the log file is generated.

```xml
<log_handler name='ni-handler' class='oracle.core.ojdl.logging.ODLHandlerFactory'>
  <property name='path' value='${domain.home}/servers/${weblogic.Name}/logs/ni-weblogic.log'/>
  <property name='maxFileSize' value='10485760'/>
  <property name='maxLogSize' value='104857600'/>
</log_handler>
```

### Table 29–1 (Cont.) Network Integrity Error Message, Classification, and Error Condition

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Classification</th>
<th>Error Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The MIB directory &quot;[mib directory]&quot; either does not exist or is not accessible. Either create this directory or change the configured MIB Directory in the Network Integrity Preferences Page (Preferences then Oracle Design Studio then Network Integrity)</td>
<td>Error</td>
<td>Occurs when the configured MIB Directory as specified in the Network Integrity Preferences Page is not accessible.</td>
</tr>
<tr>
<td>The following error occurred loading MIB &quot;[mib filename]:\n\n[error details]&quot;</td>
<td>Error</td>
<td>Occurs when the target MIB File is corrupt.</td>
</tr>
<tr>
<td>Selected node: [oid], is not readable, only readable nodes are supported.</td>
<td>Error</td>
<td>Occurs when attempting to load an OID, which is not readable.</td>
</tr>
<tr>
<td>Selected node: [oid], is not supported (only scalar and table column are supported).</td>
<td>Error</td>
<td>Occurs when attempting to load an OID, which is not scalar or a table column.</td>
</tr>
</tbody>
</table>
b. This fragment goes inside the `<loggers>` block (at the end) and defines the logger name. This name refers to the Java package of a customer’s implementation code, the log level and the handler. The handler must match the value configured in step a (for example, `ni-handler`). If necessary, tailor the log level. Consult Table 29–2 that maps the Java log levels to the ODL log levels (for example, `TRACE:32`) used in the logging.xml file.

```xml
<logger name="oracle.communications.integrity" level="TRACE:32">
    <handler name="ni-handler"/>
</logger>
<logger name="oracle.communications.activation" level="TRACE:32">
    <handler name="ni-handler"/>
</logger>
<logger name="oracle.communications.inventory" level="TRACE:32">
    <handler name="ni-handler"/>
</logger>
```

2. Save the changes to the logging.xml file.

When determining what level to set in the `logging.xml` (step 1) use Table 29–2 to map the Java Log Levels to ODL Log Levels.

<table>
<thead>
<tr>
<th>Java Log Level</th>
<th>ODL Message Type:Log Level</th>
<th>ODL Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEVERE.intValue()+100</td>
<td>INTERNAL_ERROR:1</td>
<td>The program has experienced an error for some internal or unexpected non-recoverable exception.</td>
</tr>
<tr>
<td>SEVERE</td>
<td>ERROR:1</td>
<td>A problem requiring attention from the system administrator has occurred.</td>
</tr>
<tr>
<td>WARNING</td>
<td>WARNING:1</td>
<td>An action occurred or a condition was discovered that should be reviewed and may require action before an error occurs.</td>
</tr>
<tr>
<td>INFO</td>
<td>NOTIFICATION:1</td>
<td>A report of a normal action or event. This could be a user operation, such as “login completed” or an automatic operation such as a log file rotation.</td>
</tr>
<tr>
<td>CONFIG</td>
<td>NOTIFICATION:16</td>
<td>A configuration-related message or problem.</td>
</tr>
<tr>
<td>FINE</td>
<td>TRACE:1</td>
<td>A trace or debug message used for debugging or performance monitoring. Typically contains detailed event data.</td>
</tr>
<tr>
<td>FINER</td>
<td>TRACE:16</td>
<td>A fairly detailed trace or debug message.</td>
</tr>
<tr>
<td>FINEST</td>
<td>TRACE:32</td>
<td>A highly detailed trace or debug message.</td>
</tr>
</tbody>
</table>

For more information on ODL visit

[http://download.oracle.com/docs/cd/B31017_01/web.1013/b28952/logging.htm#JIDEV400](http://download.oracle.com/docs/cd/B31017_01/web.1013/b28952/logging.htm#JIDEV400)
This chapter provides overview information about the J2EE Connector Architecture (JCA) SNMP resource adapter included with Network Integrity and other third party or customized JCA resource adapters that may be used with Network Integrity.

About Resource Adapters

A JCA resource adapter is a system-level software driver used by a Java application to connect to an Enterprise Information System (EIS). The resource adapter can be configured to use any protocol required by the EIS for connectivity. The resource adapter plugs into an application server (for example Oracle Fusion Middleware) and provides connectivity between an EIS (for example, a database system), the application server, and the enterprise application (see Figure 30–1).

JCA defined a standard architecture for connecting a J2EE platform to heterogeneous EISs. Examples of EISs include Enterprise Resource Planning (ERP) and mainframe transaction processing (TP). The connector architecture defines a Common Client Interface (CCI) for EIS access. The CCI defines a client API for interacting with heterogeneous EISs and enables an EIS vendor to provide a standard resource adapter for its EIS.

An application server that support JCA, like Fusion Middleware, can ensure seamless connectivity to multiple EISs. In the same way, any EIS with a JCA resource adapter can plug into an application server that supports JCA.

For details about the JCA 1.5 specification and additional JCA documentation, see:

http://www.oracle.com/technetwork/java/index.html
Understanding JCA Resource Adapter Connectivity Options

A resource adapter provides the following types of connectivity between an application and an EIS.

- **Outbound communication**: The resource adapter allows an application to connect to an EIS system and perform work. The application initiates all communication. The resource adapter serves as a passive library for connecting to an EIS, and executes in the context of the application threads.

- **Inbound communication**: The resource adapter allows an EIS to call application components and perform work. The EIS initiates all communication. The resource adapter can request threads from the application server or create its own threads.

- **Bi-directional communication**: The resource adapter supports both outbound and inbound communication.

Understanding JCA Resource Adapters with Network Integrity

This chapter describes productized SNMP JCA resource adapter and 3rd party or customized JCA resource adapters, and their use within Network Integrity.

Network Integrity administrators can configure the productized SNMP JCA resource adapter included with the Network Integrity software. Network Integrity system integrators can extend this SNMP JCA resource adapter with additional MIB files at run time to poll additional SNMP object identifiers (OIDs).

In addition to the productized JCA resource adapter for use with SNMP, Network Integrity system integrators can also use any standard J2EE JCA resource adapters (3rd party or customized) in their customized Network Integrity Cartridge. They can deploy these resource adapters wherever the Network Integrity application is deployed. These adapters can be standalone, or clustered within a Weblogic server.
Network Integrity Cartridges can:

- use a deployed resource adapter
- communicate with various network devices
- send commands
- collect data through various protocols (for example, SNMP, TLI, or CORBA)

See Design Studio Developer Guide for details on creating a Network Integrity Cartridge. See Network Integrity Installation Guide for details on deploying an SNMP JCA resource adapter.

About Productized SNMP JCA Resource Adapter

The SNMP discovery processor uses the SNMP JCA resource adapter, contained in the Network Integrity software to poll the SNMP enabled network devices.

The SNMP JCA resource adapter implements the connector architecture to provide SNMP functions for Network Integrity. Oracle Fusion Middleware (the application server) is the container for the SNMP JCA resource adapter and provides connection pool management. The SNMP JCA resource adapter provides outbound communication only to Enterprise Information Systems (network devices) and transaction management is not required.

The SNMP JCA resource adapter supports all SNMP-enabled network devices provided a proper set of MIB files are installed.

SNMP JCA resource adapter has record and playback functions for user who want to collect and view raw SNMP data and later reuse the data for testing purposes. For details on how to configure the SNMP resource adapter to run in record and playback mode, see “Record and Playback Mode”.

Installing the SNMP JCA Resource Adapter

The SNMP resource adapter installs as part of the Network Integrity Installer. See Network Integrity Installation Guide for more details.

Extending the SNMP JCA Resource Adapter

The SNMP resource adapter is installed with the following pre-bundled MIB files:

- ATM-MIB
- ATM-TC-MIB
- CISCO-CONFIG-MAN-MIB
- CISCO-ENTITY-VENDORTYPE-OID-MIB
- CISCO-FRAME-RELAY-MIB
- CISCO-PRODUCTS-MIB
- CISCO-SMI
- CISCO-TC
- CISCO-VLAN-IFTABLE-RELATIONSHIP-MIB
- CISCO-VTP-MIB
- ENTITY-MIB
If a device is not supported by the MIB files included with the SNMP JCA resource adapter, then the user must install additional MIB file(s) that support such a device. These additional MIB files provide the corresponding MIB OIDs and definitions for the device that the user wants to poll. Ensure that the same MIB file(s) are available in Design Studio for the corresponding cartridge development. The MIB file(s) on both Studio and the SNMP JCA resource adapter must match. Manually copy these MIB files to the SNMP JCA resource adapter.

To copy new MIB files to the SNMP JCA resource adapter, use the following steps:

1. Log in to the server where Network Integrity is installed.
2. Go to directory `<NI_HOME>/integrity/snmpAdapter/mibs`. `<NI_HOME>` is the location chosen using the NI installer during the Network Integrity installation.
3. Copy the new MIB files to this directory.

   **Tip:** There is no need to restart the server. The SNMP JCA resource adapter automatically loads the new MIB files when needed.

4. Perform an update operation of `snmpadapter` application in Admin console.

**Record and Playback Mode**

SNMP JCA resource adapter supports record and playback mode.

When the SNMP JCA resource adapter is configured to run in record mode, the resource adapter polls a network device, and the device returns the polled data to the resource adapter. The SNMP JCA adapter then returns the SNMP data to the discovery cartridge and also writes the SNMP data to a file that it stores on a local hard drive.

When the SNMP JCA resource adapter is configured to run in playback mode, the resource adapter does not require a connection to the network device. Instead the resource adapter reads the SNMP data file (created in Record mode and stored on the local hard drive) and sends the SNMP data back to discovery cartridge.

To switch the mode of SNMP resource adapter, use the following steps to create a configuration file.
1. Log in to the server where Network Integrity is installed.
2. Go to directory `<NI_DOMAIN_HOME>/config`.
3. Create a directory called `snmpAdapterConfig`.
4. Within the new directory, create a file called `snmpAdapter.properties`.
5. Add the following content to the file:

```properties
#MODE=normal
MODE=record
#MODE=playback
```

**Tip:** Enable a mode by removing the comment symbol (#) from the beginning of the line. In the above example, record mode is enabled.

The SNMP JCA resource adapter creates the record files in `NI_Domain/snmpData`. The exact directory and filename depends on the IP address. For example, device 10.156.66.191 is stored at `NI_Domain/snmpData/10/156/66/191/10.156.66.191_XXXXX.rec`, where XXXX is the name of the request set by the scan element.

Playback mode loads recorded SNMP results and sends them back to the Network Integrity cartridge without actually polling the network devices.

There is no need to restart the Weblogic server after changing the SNMP resource adapter properties file. SNMP JCA resource adapter dynamically switches the mode based on the current configuration in the properties file.

For a clustered environment, the user manually creates and modifies the properties file for every SNMP JCA resource adapter installed on every node.

**Invoking the SNMP JCA Resource Adapter in a Network Integrity Cartridge**

Design Studio creates (code-generates) the complete implementation of the SNMP processor for discovery action. This SNMP Processor can perform SNMP discoveries of SNMP enabled network devices.

After the SNMP processor discovers a device, the processor can use the SNMP JCA resource adapter to perform SNMP polling on the discovered network devices.

There is no coding effort to use the SNMP resource adapter in a Network Integrity cartridge.

**About Third Party or Customized JCA Resource Adapters**

The following sections provide information on building JCA resource adapters and on invoking third party or custom Resource adapters.

**Building a JCA Resource Adapter in Weblogic**

To create a JCA resource adapter for use in a customized Network Integrity cartridge, see:


This Fusion Middleware document provides detailed instructions for creating a resource adapter in Weblogic.
Invoking a Third Party or Customized JCA Resource Adapter

The following workflow describes the steps required to implement third party or customized JCA resource adapters in Network Integrity.

1. Deploy third party or customized JCA resource adapters into the Network Integrity system.

2. Implement a Design Studio discovery processor to invoke the third party or customized JCA resource adapter.
   a. Locate the following code auto-generated from the discovery processor.
      ```java
      @Override
      public SampleProcessorResponse invoke(DiscoveryProcessorContext context
          SampleProcessorRequest request) throws ProcessorException {
          // TODO Auto-generated method stub
          return null;
      }
      ``
   b. Use the SampleProcessorRequest generated class to obtain the address scope, property group, and other attributes.

   **Tip:** This class provides important elements used when invoking a resource adapter. For example, to use a TL1 resource adapter to make a TL1 request, the TL1 resource adapter needs to know which device it should communicate with. This information is obtained from the SampleProcessorRequest in the following sources:
   - IP address: available from the address scope
   - port number: available from the property group
   - login information for the TL1 session including username and password: available from the property group

c. Use the data provided by SampleProcessorRequest to implement the Java code to invoke the JCA resource adapter.

Depending on the resource adapter, the way to invoke a resource adapter can different. Typically the invoke process requires several JNDI name lookups to get some JCA Connection Factory and Interaction Specification classes. From the JCA Connection Factory, the user can create Interaction. Next is to do the execution from Interaction by passing the Interaction Specification.

If user is using an existing 3rd party resource adapter, it should come with a developer guide that provides the detailed instruction on how to implement the client code to invoke this resource adapter. If a user creates a customized resource adapter from scratch, the user should have all the knowledge on how to invoke this customized JCA resource adapter.

The following code snippet demonstrates how to invoke a JCA resource adapter that implements Common Client Interface (CCI):

```java
context = new InitialContext();
SampleAdapterConnectionSpecImpl cspec =
    (SampleAdapterConnectionSpecImpl)context.
lookup(JNDI_SAMPLE_CONN_SPEC);
cxFactory = (ConnectionFactory) context.lookup(JNDI_SAMPLE_CONN_FACTORY);
connection = cxFactory.getConnection(cspec);
ispec = (SampleAdapterInteractionSpec)context.lookup(JNDI_SAMPLE_INTER_SPEC);
```
interaction = connection.createInteraction();
RecordFactory recordFactory = cxFactory.getRecordFactory();
IndexedRecord input = recordFactory.
createIndexedRecord(SampleAdapterIndexedRecord.OUTPUT);
input.add(request);
IndexedRecord output = recordFactory.
createIndexedRecord(SampleAdapterIndexedRecord.OUTPUT);
interaction.execute(ispec, input, output);
out=(String)output.get(SampleAdapterIndexedRecord.MESSAGE_FIELD);

In this example, the “out” contains the collected results as an XML document as String. However, different resource adapter have different output. To detail all possible kinds of output is beyond the scope of this document.

The final output should be wrapped inside the SampleProcessorResponse class (code-generated) and return as the returned value of this invoke method.
This chapter provides overview information about the Oracle Business Intelligence (BI) Publisher and reports extensibility for Oracle Communications Network Integrity.

About BI Publisher

Oracle BI Publisher (formerly XML Publisher) is an enterprise reporting solution to design, use, author, manage, and deliver report documents in various formats. This tool provides the following services:

■ Creating New Report
■ Uploading Existing Report
■ Organizing reports in folders
■ Viewing a report in various formats like RTF, HTTP, PDF, XLS
■ Scheduling a report for delivery to destination like Email id
■ Internationalization of report
■ Configuring various data sources like Oracle Business Intelligence Enterprise Edition (OBIEE), web service, and database

This chapter does not describe all Oracle BI Publisher features. Customized, or more advanced reports can be created using BI Publisher. For more information about Oracle BI Publisher, see the Oracle Business Intelligence Publisher Installation documentation:

http://download.oracle.com/docs/cd/E12844_01/doc/bip.1013/e12690/toc.htm

About BI Publisher Desktop

BI Publisher Desktop is a plug-in for Microsoft Word that enables a user to create report templates and represent data from data sources. This plug-in provides various wizards to create tables, grouped tables, graphs, charts, and so on. It internally uses Microsoft Word’s native features and the eXtensible Stylesheet Language (XSL) to implement these wizards.

Reports Provided with Network Integrity

Network Integrity includes the following BI Publisher reports:

■ Scan History Report
■ Discovery Scan Summary Report
Reports Provided with Network Integrity

- Device Discrepancy Detection Summary Report
- Device Discrepancy Detection Detail Report
- Discrepancy Corrective Action Report

Scan History Report

The **Scan History Report** shows the discovery and discrepancy summaries for each scan for each scan configuration falling within the specified start and end dates. This report is accompanied by the following graphs:

- Discovery Scan History: a graph showing a history of the run discovery scans.
- Discrepancy Scan History: a graph showing a history of the run discrepancy scans.
- Discrepancy Severity History: a graph showing a history of the discrepancies by severity.

The following fields are used to generate this report:

- Start Time: the date stamp indicating when a scan started.
- End Time: the date stamp indicating when a scan finished.

Discovery Scan Summary Report

The **Discovery Scan Summary Report** shows the summary of the latest scan for each scan configuration, per vendor and per device type. This report generates a pie-chart, illustrating the summary findings, for each scan configuration.

The following fields are used to generate this report:

- Vendor: the name of the vendor for the discovered device.
- Device Type: the type of device discovered.

Device Discrepancy Detection Summary Report

The **Device Discrepancy Detection Summary Report** shows the summary of the latest scan for each scan configuration. This report generates a pie-chart that shows the accuracy of the latest scans for each scan configuration.

The following fields are used to generate this report:

- Vendor: the name of the vendor for the discovered device.
- Device Type: the type of device discovered.

Device Discrepancy Detection Detail Report

The **Device Discrepancy Detection Detail Report** lists details of all discrepancies for the latest scan for each scan configuration.

The following fields are used to generate this report:

- Vendor: the name of the vendor for the discovered device.
- Root Device Name: the name of the root device in the scan result tree.
- Root Device Type: the type of the root device in the scan result tree.
- Owner: the user name of the owner of the discrepancy.
- Parent Entity Type: the type of the parent entity on which discrepancy occurred.
Discrepancy Corrective Action Report

The Discrepancy Corrective Action Report shows corrective actions against specified discrepancies for the latest scan for each scan configuration. Discrepancies that have not been actioned are not considered in this report.

The following fields are used to generate this report:

- Submitted By: the user who submitted the discrepancy for correction.
- Action: the action taken against the discrepancy.
- Discrepancy Status: Status of the discrepancy.
- Owner: the user name of the owner of the discrepancy.
- Priority: the priority of the discrepancy.
- Failure Reason: the reason for failure for the corrected discrepancy.
- Discrepancy Type: the type discrepancy (for example, entity+, entity-, attribute)
- Entity Type: the type of the entity on which discrepancy occurred.
- Inventory Value: the value of the field on the inventory side on which discrepancy occurred.
- Network Value: the value of the field on the network side on which discrepancy occurred.

Network Integrity Report Building Blocks

Network Integrity uses BI Publisher to generate reports. These reports use Rich Text Format (RTF) templates and a report Definition Extensible Markup Language (XML) file (.xdo) that tracks the various data sources, parameters, values, rendering options, and report properties that BI publisher uses to populate the RTF reports.

RTF Templates

The RTF template defines the layout and display for report data. The BI Publisher server uses the templates to format the data from various data sources. These formats include tables, charts, graphs, and so on.
A user can use general word processing features and BI Publisher’s simplified tags for XSL expressions. The template parser inside the server processes the RTF document and converts it to XLS formatting objects (XSL-FO), useful for rendering the final formatted report in PDF, RTF, or HTML formats.

The details of creating templates are covered in next chapters.

**Report Definition Files**

The reports Definition file (.xdo) contains all the configurations pertaining to a report and all the layout definitions that refer to the various templates used by the report.

**Data Source Query Tools**

The data source tools use various protocols such as SQL, web service, and XML files that can get data from data sources for use in a report. Network Integrity typically uses SQL queries pertaining to Network Integrity schemas and retrieves query results from a Network Integrity database through Java database connectivity (JDBC).

See *Oracle Communications Information Model Reference* and *Network Integrity Information Model Reference* for more information about how entities and parameters relate to the Network Integrity database.

**Parameters**

Parameters capture values at run-time and use these values to filter or manipulate data. Parameter definitions includes data type, default value, parameter type, display label, and list of values.

**List of Values**

The Menu parameter type uses the list of values definitions that BI Publisher render as a dropdown in a report. The dropdown menu is populated with values defined in the list of values. This definition can be a pre-defined hard coded list or it can be the result of an SQL query.

**Rendering Options**

You can render reports using formats like PDF, RTF, HTML, EXCEL, and so on. You can choose the render option for the reports they create.

**Report Properties**

The following list provides details on various report property functions:

- **Run report online**: Enable this property to view a report online. You can only schedule and view reports in history after BI Publisher runs the scheduled report.

- **Show Controls**: Enable this property to see the parameters and other View report screen controls like View, Send, Schedule, Analyzer, and so on.

- **Show Report Links**: Enable this property to see a link menu in the view report screen that contains different menu items that provide HTTP links for a report. You can bookmark these links.

- **Auto Run**: Enable this property to cause a report to run automatically when a user attempts to view it. If this property is not enabled, the user must click View in the view screen of the report.
■ **Enable document cache**: Enable this property to provide a cache in the report document. This property is appropriate for reports that connect to live databases that are frequently updated.

### Developing BI Publisher Reports

This section explains how to develop a custom report by creating an example report called *Discrepancy Severity History Report*.

#### Report Requirements

The *Discrepancy Severity History Report* requires the following:

- Display the following fields:
  - Scan Name
  - Discrepancy Scan Start & End time
  - Counts of Critical, Major, Minor, Warning discrepancy types
- Filter on Discrepancy Start Date.
- Group data by Scan Name.
- Create a bar chart for all discrepancy severity types.

The following sections describe how to run the *Discrepancy Severity History Report*. Knowledge of the Network Integrity data model is necessary to write the SQL queries needed for the *Discrepancy Severity History Report*. See *Oracle Communications Information Model Reference* and *Network Integrity Information Model Reference* for information about the data model.

#### Configuring a Data Source Tutorial

The following steps provide a tutorial for configuring a data source:

1. Log on to BI Publisher as an Administrator.
2. Click the Admin tab.
3. Click JDBC Connection under the Data Sources section.
4. Click demo to edit the connection details for this data source.
5. In the Update Data Source screen, enter the following details:
   a. For Connection String, enter:
      ```
jdbc:oracle:thin:@<Host name>:<Port>:<SID>
      ```
      For example, a sample connection string may look like
      ```
jdbc:oracle:thin:@myhost:1521:orcl
      ```
   b. For the User Name, enter:
      ```
      oe
      ```
   c. For the Password, enter:
      ```
      oe
      ```
   d. For the Database Driver Class enter the default:
      ```
      oracle.jdbc.driver.OracleDriver
      ```
6. Click Test Connection.
Developing BI Publisher Reports

7. Click **Apply** to save the connection details after confirmation that the connection is successfully established.

**Creating a Report Tutorial**

The following steps provide a tutorial for creating a report:

1. Log on to BI Publisher as an Administrator.
2. Click the **Reports** tab.
3. Click the **MyFolders** link.
4. Click **Create a new folder link** in the **Folder and Report Tasks** panel.
5. Enter the following folder name:
   - **Test**
6. Click **Create** to create the folder.
7. Click the new **Test** folder.
8. Click **Create a new Report**.
9. Enter the following report name:
   - **Discrepancy Severity History Report**
10. Click **Create** to create the report.
11. Click **Edit** link below the report name.
12. Select **Parameter**.
13. Click **New** button.
14. Enter the following values to create a parameter called **p_StartDate**.
   a. For **Identifier**, enter:
      - **p_StartDate@**
   b. For the **Parameter Type**, select Date.
   c. For the **Display Label**, enter:
      - **Discrepancy Begin Date Range Start**
   d. For the **Date Format String** enter:
      - **MM-dd-yyyy**
15. Repeat steps 12 to 14 to create a parameter called **p_EndDate**. Label the **Identifier** as **p_EndDate** and the **Display Label** as **Discrepancy Start Date Range End**.
16. Select **Data Model**.
17. Click **New**.
18. Enter the following values to create a data set called **MyDataSet**.
   a. For **Name** enter:
      - **MyDataSet**
   b. For **Type**, select **SQL Query**.
   c. For **Data Source** select **demo** from the dropdown menu.
Developing BI Publisher Reports

Network Integrity Reports Extensibility

For the SQL Query text box enter the following SQL query as required by the Network Integrity model to display the field described in "Report Requirements":

```sql
select dc1.NAME as SCANCONFIGNAME,
        case when dsr.DISCREPDETECTBEGINTIME is null then null else
            to_char(dsr.DISCREPDETECTBEGINTIME,'YYYY-MM-DD HH12:MI:SS TZR AM') end STARTTIME,
        nvl(to_char(dsr.DISCREPDETECTENDTIME, 'YYYY-MM-DD HH12:MI:SS TZR AM'), '') as DISCREPDETECTENDTIME,
        CASE WHEN DISCREPDETECTBEGINTIME IS NULL THEN NULL ELSE nvl(ddc.NUMBERCRITICAL,0) END C,
        CASE WHEN DISCREPDETECTBEGINTIME IS NULL THEN NULL ELSE nvl(ddc.NUMBERMAJOR,0) END M,
        CASE WHEN DISCREPDETECTBEGINTIME IS NULL THEN NULL ELSE nvl(ddc.NUMBERMINOR,0) END mi,
        CASE WHEN DISCREPDETECTBEGINTIME IS NULL THEN NULL ELSE nvl(ddc.NUMBERWARNING,0) END w,
from DISCONFIG dc left outer join DISCONFIG dc1 on dc.scanconfig= dc1.entityid
        left outer join DISSCANRUN dsr on dc.ENTITYID=dsr.CONFIG
        left outer join DISDISCREPANCYCOUNTS ddc on dsr.DISCREPANCYCOUNTS=ddc.ENTITYID
where dc.scanconfig is not null and
      (:p_StartDate is null or trunc(dsr.DISCREPDETECTBEGINTIME, 'DDD') >= :p_StartDate) and
      (:p_EndDate is null or trunc(dsr.DISCREPDETECTBEGINTIME, 'DDD') <= :p_EndDate)
```

19. Click Save to save the data set. The XML source data can now be viewed in a report.

Building an RTF Template Tutorial

The following steps provide a tutorial for building an RTF template:

1. Using Microsoft Word, create an RTFF file called DiscrepancySeverityHistory.rtf.
2. Open the Microsoft Word file.
3. From the AddIns menu, select BI Publisher.
4. Log on to BI Publisher as an Administrator. The Login Wizard immediately opens to a screen showing the DiscrepancySeverityHistory.rtf report selected.
5. Click the Open Report button.
6. Enter the following text as the title of the report (see Figure 31–1):
   Discrepancy Severity History Report
7. Enter the following text as a description of the report (see Figure 31–1):
   The report shows historical discrepancies severity summaries for scans.
8. Enter the following code in the field after the description (see Figure 31–1). This code is a declaration of the parameter strings already defined in "Creating a Report Tutorial":

```xml
<?param@begin:p_Start;'None'?>
```
9. Below the declaration, insert a table with two columns (see Figure 31–1).

10. Enter the table column titles as FilterName and FilterValue and enter StartDate and EndDate in first column of both the rows (see Figure 31–1).

11. Insert the following code into the second and third rows of the second column respectively (see Figure 31–1).

```html
<?$p_Start?>
<?$p_End?>
```

**Figure 31–1  Report Example**

![Discrepancy Severity History Report](image)

12. Select Oracle BI Publisher then Insert then Table wizard.

13. Select the Table option.

14. Click Next.

15. Select the data set that you defined in "Creating a Report Tutorial".

16. Select the field to display on the report and arrange them in an appropriate order.

17. Click Next.

18. Select Scanconfigname from the Group By dropdown menu and to be grouped by horizontally.

19. Click Next.

20. Select any field from the Sort By dropdown menus. This step is optional.

21. Label each field as required.

22. Click Finish.

23. Place the cursor below the table within the groupby clause.

24. Select Oracle BI Publisher then Insert then Chart.

25. From the Data explorer panel, drag the STARTTIME field into the text box adjacent to Labels.

26. Drag the C, M, Mi, W fields into the text boxes adjacent to Values.

27. From the Type dropdown menu, select Line Graph.

28. From the Style dropdown menu, select Confetti.
29. Select the **Group Data** check mark.

30. Select the **Chart is inside group** check mark.

31. Change the items as required in the **Properties** list. The values are **True** or **False**.

32. Click **OK** to insert the chart into the Microsoft Word document.

33. Select **Oracle BI Publisher** menu then **Upload Template As** to upload the template to the BI Publisher server.

34. Enter a name for the new template, for example, **SevHistoryReportTemplate**.

35. Click **OK**.

36. Navigate to the BI Publisher server where the template is now located.

37. Click **View** to see the reports.

**Using Microsoft Word Native Features and XSL**

RTF templates can use standard word processing features of Microsoft Word like tables, formatting, styles, and so on. The BI Publisher Desktop plug-in also provides additional wizards for use in Microsoft Word. These wizards internally generate XSL equivalent code which is interpreted by BI Publisher server while rendering the template.

**Using BI Publisher Features**

Oracle BI Publisher provides the following important features:

- Scheduling Reports
- Localizing Reports

**Scheduling Reports**

To schedule reports, use the following tasks:

- **Enabling BI Publisher Scheduler Tutorial**
- **Using BI Publisher to Schedule Reports Tutorial**

**Enabling BI Publisher Scheduler Tutorial**

BI Publisher requires a database to create a scheduling schema and a database user to use the scheduling schema.

To create an database user for scheduler schema and grant required permissions to that user, use the following steps:

1. Use the following SQL example to create a database user:

   ```sql
   SQL> CREATE USER bipubsched
   2   IDENTIFIED BY welcome
   3   DEFAULT TABLESPACE USERS
   4   TEMPORARY TABLESPACE TEMP
   5   QUOTA 20G ON USERS
   6   QUOTA 1M ON TEMP;
   
   User created.
   
   SQL> GRANT CREATE SESSION TO bipuser; -- or "GRANT CONNECT TO bipuser;"
   ```
Grant succeeded.

SQL> grant create table to bipublisher;
Grant succeeded.

2. Log on to BI Publisher as an Administrator.
3. Select Admin.
5. Enter the following values:
   a. For Database Connection Type dropdown menu, select jdbc.
   b. For Database Type dropdown menu, select Oracle 10g.
   c. For Connection String, enter jdbc:oracle:thin:@10.177.219.0:1521:ORCL.
   d. For Username, enter bipudsched.
   e. Enter a Password.
   f. For Database Driver Class, enter oracle.jdbc.OracleDriver.
6. Click Text Connections to verify if the connection is successful.
7. If the connection was successful, click Install Schema to install the schema in the database.
8. Restart BI Publisher to apply the change. You can schedule reports after the server restarts.

Using BI Publisher to Schedule Reports Tutorial
To schedule reports, use the follow procedure:
1. In BI Publisher, navigate to the folder you stored your reports.
2. Click Schedule below the report you want to schedule.
3. In the Schedule Reports screen, enter values as required.
   
   **Tip:** Before you can configure Email destinations, ensure that the mail server has been configured. This can be done from the Admin tab on the main menu in the Delivery section.

4. Click Submit to schedule the report. When the scheduled report has been run, the report is stored in the History.

Localizing Reports
The localization of a report is achieved by translating the templates used for that report and adding those templates to the report Definition in BI Publisher. There are two options for adding translated templates to a report Definition.

Localized Template Option
In this option, there is one RTF template for one locale supported for the report. There are $n$ RTF templates for $n$ locales supported. Figure 31–2 illustrated that there is an RTF template with French locale and the data is in an XML file. BI Publisher uses both and renders the report in French text.
Naming convention for the report template is `TemplateName_<language code>_<TERRITORY CODE>.rtf`.

This RTF template must be uploaded to BI Publisher to include this in the report Definition.

**XLIFF File Option**

In this option, there is one RTF template for one locale supported for the report. There are `n` RTF templates for `n` locales supported. Illustrated that there is an RTF template with French locale and the data is in an XML file. BI Publisher uses both and renders the report in French text.

In this option, there one RTF template but `n` XML localization interchange file format (XLIFF) files for `n` locales supported. *Figure 31–3* illustrates that BI Publisher receives data in XML, base template, and an XLF file with translated text strings to render the report in French text.

Naming convention for the report template is `TemplateName_<language code>_<TERRITORY CODE>.rtf`.

Upload all supported `n` XLF files and corresponding `n` locales to the BI Publisher report to include these into the report Definition.
Installing and Integrating BI Publisher with Network Integrity

BI Publisher can be installed as a standalone application or as a web application deployed in the same application server where the Network Integrity is deployed.

BI Publisher is partially integrated with Network Integrity which supports various BI Publisher application links in the Network Integrity Links panel. The BI Publisher application can be launched from Network Integrity using the BI Publisher link.

Installing BI Publisher in Standalone Mode

The following link provides the steps to install BI Publisher in standalone mode:

http://download.oracle.com/docs/cd/E12844_01/doc/bip.1013/e12690/T434820T487782.htm

Deploying BI Publisher

To deploy BI Publisher as an Enterprise ARchive (EAR) in a WebLogic application server where Network Integrity is running, use the following steps:

1. Download the BI Publisher zip file.
2. Unzip the zip file.
3. Navigate to the RHLinux/Oracle_Business_Intelligence_Publisher_Standalone/manual/generic directory where the xmlpserver war and ear files are located.
4. Explode the war file to create an xmlpserver directory with all the war files present within using the following command:
   ```
   jar -xvf xmlpserver.war
   ```
5. Deploy the xmlpserver folder onto WebLogic as an application.
6. Copy all the files under RH Linux/Oracle_Business_Intelligence_Publisher_Standalone/manual/generic directory into (for example) ~/BIP or /home/<username>/BIP
7. Search the xmlp-server-config.xml file in the Network Integrity domain of WebLogic.
8. Insert the following in the file:
   ```
   path="/home/<username>/BI
   ```
9. Save the file.
10. Copy font files from RH Linux/Oracle_Business_Intelligence_Publisher_Standalone/manual//fonts and put them in jdk1.6_011/jre/lib/fonts. This is the JDK that the Network Integrity installer uses.
11. Restart the application server.
13. Login as administrator.

Integrating BI Publisher with Network Integrity using WebLogic Enterprise Manager

Complete one of the following prerequisites before attempting this procedure:

- Installing BI Publisher in Standalone Mode
Installing and Integrating BI Publisher with Network Integrity

- **Deploying BI Publisher**

To integrate BI Publisher with Network Integrity using the WebLogic Enterprise Manager, use the following steps:

2. Log on to Enterprise Manager using the username and password used to log on to WebLogic Admin Console.
3. Click the WebLogic Domain folder link in the left side navigation panel.
4. Expand the Weblogic Domain folder to display the domains.
5. Select the domain where the Network Integrity application is installed.
6. Select **System Mbean Browser** from the dropdown menu. The System Mbeans shows all the Mbeans deployed in WebLogic server.
7. Select the NIRegionLinksService Network Integrity Mbean to view its properties.
8. Enter BI Publisher link against any URL property (for example, URL06).
9. Enter the display string of the URL against URLName06. This link appears in the **Links** panel of the Network Integrity user interface the next time the user interface refreshes.

Integrating BI Publisher with Network Integrity using JConsole

Complete one of the following prerequisites before attempting this procedure:

- **Installing BI Publisher in Standalone Mode**
- **Deploying BI Publisher**

To integrate BI Publisher with Network Integrity using the JConsole, use the following steps:

1. Use the following command to launch JConsole in Windows.

```
%JAVA_HOME%/bin/jconsole -Djava.class.path=%JAVA_HOME%/lib/jconsole.jar;%JAVA_HOME%/lib/tools.jar;%WLS_HOME%/server/lib/wljmxclient.jar -Djmx.remote.protocol.provider.pkgs=weblogic.management.remote
```

2. Use the following command to launch JConsole in Linux.

```
jconsole -Djava.class.path=$JAVA_HOME/lib/jconsole.jar:$JAVA_HOME/lib/tools.jar:$WLS_HOME/server/lib/weblogic.jar -Djmx.remote.protocol.provider.pkgs=weblogic.management.remote
```

3. In the JConsole **New Connection** screen, select the **Remote Process** option.

   The host name corresponds to the IP address of the host where Network Integrity is installed.

   The port number is the port on which the managed server is running. For example, 7003.

5. Enter the same **username** and **password** used for accessing the WebLogic admin console and click **connect**.
6. Navigate to `oracle.communications.integrity` then `NIRegionalLinksService` then `oracle.communications.integrity` then `Attributes`. `NIRegionalLinkService` is the Mbean placeholder for BI Publisher link.

7. Set URL06 with the BI Publisher WebLogic URL.

8. Set URLName06 with the corresponding display title. Now the Network Integrity shows BI Publisher link that points to the BI Publisher URL in the links panel after next page refresh.

**Integrating BI Publisher with the Network Integrity Installer**

Complete one of the following prerequisites before attempting this procedure:

- Installing BI Publisher in Standalone Mode
- Deploying BI Publisher

To integrate BI Publisher with Network Integrity using the Network Integrity Installer, use the following steps:

1. Log on to Network Integrity.
2. Navigate to the Network Integrity Installer screen.
3. Navigate to the System Wide Shortcut links (Optional) screen.
4. Enter a name for the link to BI Publisher in the Name text box (for example, BI Publisher).
5. Enter the URL for the link to BI Publisher in the Link text box.
6. Click Next and follow the Wizard instructions.

**Uploading and Exporting Reports**

Oracle BI Publisher provides the following methods to upload and export reports:

- Exporting a Report
- Uploading a Report

**Exporting a Report**

To export a BI Publisher report, use the following steps:

1. Log on to BI Publisher as administrator.
2. Click the Admin tab.
3. In the System Maintenance section, click Report Repository.
4. Change the Path if required.
5. Copy the path and paste it into the Explorer Address bar, or an equivalent software.
7. Select the Username associated to the report you want to export. There are \( n \) folders for \( n \) reports in that user folder. These reports can be zipped and stored in any location for future use.
Uploading a Report

To upload a BI Publisher report, use the following steps:

1. Log on to BI Publisher as administrator.
2. Click the Reports tab.
3. Click My Folders.
4. Click Create a new folder.
5. Click Create.
6. Click the newly created folder link.
7. Click Upload a Report.
8. Save the reports downloaded from secondary memory or from location where they were stored.
9. Unzip each of the zip files into individual folders.
10. Click browse and navigate to the unzipped folder.
11. Select the file with .xdo extension.
12. Click upload button. The report is created in you folder now.
13. Click the report – “Edit” link and click the “Layouts” link in the left side navigation.
14. Click browse and upload all the RTF files in the unzipped folder.
15. Click Save link in the left navigation.
16. Repeat steps 10 to 15 for all additional unzipped report folders.
This chapter provides overview information about Service-Oriented Architecture (SOA) extensibility for Oracle Communications Network Integrity.

About SOA Extensibility

SOA extensibility topics covered in this chapter include creating an SOA development environment, setup, development, and testing of the Network Integrity SOA application.

The Business Process Execution Language (BPEL) provides enterprises with an industry standard for business-process orchestration and execution. Using BPEL, you design a business process that integrates a series of discrete services into an end-to-end process flow.

The Oracle BPEL Process Manager is a tool for designing and running business processes. This product is used to create, deploy, and manage cross-application business processes with both automated and human workflow steps in a service-oriented architecture.

The Sample Network Integrity SOA application provides a BPEL process that contains two parallel sequences. These sequences automate search and update Network Integrity discrepancies.

The following shows how this automation occurs:

1. Search for Network Integrity discrepancies of type attribute mismatch for nativeEmsServiceState and update their resolution to Correct in UIM, if those discrepancies’ network value is In service and import value is Out of service.

2. Search for Network Integrity discrepancies of type attribute mismatch for physicalAddress and update their priority to High and discrepancy owner to given input value.

Purpose of Documentation

The developer should learn to install SOA, setup SOA Development environment, and use it for Network Integrity SOA application extensibility.

Extensibility Tasks

The tasks involve setting up of developer environment to update and extend the Network Integrity SOA application for future requirements.

Required software includes:
Extensibility Tasks

To implement SOA extensibility, use the following tasks:

- Installing Oracle Weblogic Server
- Installing Oracle JDeveloper
- Installing Oracle Application Runtime
- Installing Oracle SOA Suite
- Creating SOA Metadata Service Schemas
- Updating JDeveloper for Latest SOA Composite Editor
- Creating WebLogic Domain with SOA Products
- Creating and Updating Sample SOA Application Using Network Integrity Web Service
- Starting and Stopping SOA Servers
- Building and Deploying the SOA Application
- Testing Sample SOA application
- Testing Network Integrity SOA Application Using EM
- Testing Network Integrity SOA Application Using soa-infra
- Testing Network Integrity SOA Application Using SOAP UI Tool

Installing Oracle Weblogic Server

See Network Integrity Installation Guide for information about supported software versions.

To install Oracle Weblogic Server:

1. Download Oracle WebLogic Server.
2. Run the installation script. For example: /wls1033_linux32.bin
3. Click Next.
4. Enter the WL_Home directory location to create a home directory for Oracle Fusion Middleware.
5. Click Next.
6. Select the I wish to receive security updates via Oracle Support check box and click Next. (Optional)
7. Select **Custom** for the installation type.
8. Click **Next**.
9. Select the **WebLogic Server** check box to install all WebLogic Server components.
10. Click **Next**.
11. Select the **Sun JDK** check box.
12. Click **Next**.
13. Review the installation directories.
14. Click **Next**.
15. Review the installation summary of the products and JDKs to be installed.
16. Click **Next**. This step begins the installation.
17. When the installation is complete, deselect **Run Quickstart**.
18. Click **Done**.
19. Setup `BEA_HOME`, `JAVA_HOME`, `WL_HOME` environment variables and update `PATH` with the Java executable location. For example,

   ```
   export BEAHOME=/opt/beahome
   export WL_HOME=$BEAHOME/wlserver_10.3
   export JAVA_HOME=$BEAHOME/jdk160_14_R27.6.5-32
   export PATH=$JAVA_HOME/bin:$PATH
   ```

### Installing Oracle JDeveloper

To install Oracle JDeveloper, use the following procedure:

1. Download Oracle JDeveloper 11g R1 PS2 software from the Oracle software delivery Web Site:
   
   [https://edelivery.oracle.com/](https://edelivery.oracle.com/)

2. Unzip the installer to any directory.
3. Open a console.
4. Change the console directory to the unzipped installer directory.
5. Run the installer using the following command:

   ```
   java -jar jdevstudio11112install.jar
   ```

   The Installer starts extracting the setup files and Installation wizard opens when it reaches to 100%.
6. Click **Next**.
7. Select **Use the existing Middleware Home** to select the Middleware home you created in "Installing Oracle Weblogic Server".
8. Select **JDeveloper Studio and ADF** to install all JDeveloper Studio and ADF components.
9. Click **Next**.
10. Select the existing **Sun SDK**.
11. Click **Next**.
12. Confirm JDeveloper and WLS home directories and click Next.
13. Review the Installation summary and click Next. This step begins the installation.
14. Click Done when the installation is complete.

Installing Oracle Application Runtime

See Network Integrity Installation Guide for information about supported software versions.

To install Oracle Application Runtime:
1. Download Oracle Application Development Runtime from the Oracle software delivery Web site:
   https://edelivery.oracle.com/
2. Unzip the installer to any directory.
3. Open a console.
4. Change the console directory to the unzipped installer directory.
5. Run the installer using the following command:
   . Disk1/runInstaller
6. Enter the JAVA HOME location to launch installation wizard.
7. Click Next.
8. Click Next button after Prerequisite Checks are complete.
   **Tip:** Install the required system package if a check fails.
9. Click Browse and navigate to WL_Home.
10. Click Next.
11. Click Install.
12. Click Next after the installation is complete.
13. Click Finish.

Installing Oracle SOA Suite

To install Oracle SOA Suite, use the following procedure:
1. Download Oracle SOA Suite 11gR1 PS2 software from the Oracle software delivery Web site:
   https://edelivery.oracle.com/
2. Unzip the installer to any directory.
3. Open the console and change to unzipped folder directory.
4. Run the installer using the following command:
   . Disk1/runInstaller
5. Enter the JAVA HOME location to launch installation wizard.
6. Click Next.
7. Click Next after Prerequisite Checks are complete.
Tip: Install the required system package if a check fails.

8. Click Browse and navigate to WL_Home. Do not modify the Oracle Home Directory name.
9. Click Next.
10. Click Install.
11. Click Next after the installation is complete.
12. Click Finish.

Creating SOA Metadata Service Schemas

To create a metadata service (MDS) schema for the Business Activity Monitoring (BAM) and SOA servers, use the following procedure:

1. Download Oracle Fusion Middleware Repository Creation Utility software from the Oracle software delivery Web site:
   https://edelivery.oracle.com/
2. Unzip the Repository Creation Utility (RCU) to any directory.
3. Open the console and change to unzipped folder directory.
4. Run the installer using the following command:
   ./rcuHome/bin/rcu
5. Click Next.
6. Select Create in the Create Repository screen
7. Click Next.
8. Enter database details as required.
9. Click Next.
10. Click OK.
11. Select Create a new Prefix in the Select Components screen and enter a prefix in the text box.
12. Select the following from the Component list:
   - Metadata Service
   - SOA Infrastructure
   - Business Activity Monitoring
   - User Messaging Service
   These components are required for the SOA and BAM servers.

   Tip: Remember the Schema Owners for subsequent procedures.

13. Click Next.

   Tip: Remember the Schema Passwords for subsequent procedures.

15. Click Next.
16. Review the **Schema Owner**, **Tablespace Type**, and **Tablespace Name** for each **Component** in the **Summary** screen.

17. Click **Next** to accept the settings.

18. Click **OK** to create the tablespaces.

19. Click **OK** when the prerequisites are complete.

20. Click **Create** in the Summary screen to create the tablespaces. This step can take up to ten minutes.

21. Click **Close** after the tablespaces are created.

**Updating JDeveloper for Latest SOA Composite Editor**

SOA design time in JDeveloper requires a JDeveloper extension called SOA Composite Editor. While this is normally updated over the network when using release-level software, you can also perform the update manually if you have the extension file. JDeveloper periodically prompts you to accept an automatic network update. Since this is released software, you have the option to click **OK** to update to a newer version.

To update JDeveloper for the latest SOA composite editor, use the following procedure:

1. Start JDeveloper Studio.

2. Select Default Role.

3. Deselect **Show this dialog every time**.

4. Click **OK**.

5. Click **No** for **Migrate from previous release**. After starting JDeveloper, wait for the Integrated Weblogic Domain to be created. This domain is created the first time you run JDeveloper after installation. It is not used by SOA. Watch for the completion message for setting up the domain in the JDeveloper Messages log window at the bottom of the JDeveloper IDE:

   
   [12:37:11 PM] Creating Integrated Weblogic domain...
   [12:38:05 PM] Extending Integrated Weblogic domain...

Now you can update the SOA Composite Editor extension. These instructions show you how to update the extension over the network.

6. Select **Help | Check For Updates**.

7. Click **Next**.

8. Select **Search Update Centers**.

9. Select **Oracle Fusion Middleware Products**.

10. Click **Next**. The system searches the update center for extensions.

11. From the list of extensions, select **Oracle SOA Composite Editor**.

12. Click **Next** to begin downloading. The extension is about 230 MB and takes a few minutes to download. When the extension finishes downloading, it is listed with the version number detail.

13. Confirm that the SOA Composite Editor version matches the version number detail as shown: 11.1.1.3.0.25.57.
14. Click Finish.
15. Restart JDeveloper when prompted.
16. Click No for Migrate from previous release.
17. When JDeveloper is running again, select Help then About.
18. Confirm that the JDeveloper version is as shown: JDEVADF_11.1.1.3.
      PS2_GENERIC_100408.2356.5660
19. Select the Version tab.
20. Confirm the SOA Composite Editor was installed properly by confirming the
    version: 11.1.1.2.36.55.36 or the version you noted above.

Creating WebLogic Domain with SOA Products

To create an Oracle WebLogic domain with the required products for SOA
applications, use the following procedure:

1. Open the console and change to unzipped folder directory.
2. Run the following command:
   ./<BEAHOME>/weblogic_10.3/common/bin/config.sh
3. When the Welcome screen appears, select Create a new WebLogic domain.
4. Click Next.
5. Select Generate a domain, SOA Suite, Enterprise Manager, and Business
   Activity Monitoring. Dependent products are selected automatically.
6. Click Next.
7. Enter domain1 for the domain name.
8. Click Next.
9. Enter the user name weblogic and a password. The password welcome1 is
   assumed in this document.
10. Click Next.
11. Select Sun SDK 1.6_11 and leave Development Mode checked.
12. Click Next.
13. Select the check boxes for the components that you want to change.
14. Enter Welcome1 for the Schema Password.
15. Change the Service, Host Name, and Port values as required.
16. Click Next.
17. Review the Schema Owners for the individual component schemas and confirm
    that the owners match those selected in the "Creating SOA Metadata Service
    Schemas" procedure.

   Tip: To change the Schema Owner field, use the following steps:
   1. Remove the check boxes for all Component Schema items.
   2. Select the check box for the Component Schema that you want to change.
   3. Change the Schema Owner field.
   4. Remove the check box for the component schema item you changed.
18. Click Next to begin a data source connection test.

19. Click Next if all connection tests are successful. If the connection tests are not successful, click Previous and correct any errors.

20. Click Next.

21. Click Create in the Configuration Summary screen.

22. Click Done when the domain has been created.

When a domain is created, the Configuration Wizard creates one admin server and two managed servers with the following details:

- **Admin Server**
  - Name: admin_server
  - Port: 7001

- **SOA Server**
  - Name: soa_server1
  - Port: 8001

- **BAM Server**
  - Name: bam_server1
  - Port: 9001

Please check startManagedServer_readme.txt file in domain folder to start the servers.

**Creating and Updating Sample SOA Application Using Network Integrity Web Service**

To update an SOA application using the Network Integrity SOA application, use the following procedure:

1. Download the Sample Network Integrity SOA application (NetworkIntegrity-SOA_Sample_App-version.zip) software from the Oracle software delivery Web site:

   https://edelivery.oracle.com/

2. Unzip the application to any directory.


4. From the Jdeveloper main menu, choose File then Open then browse to NISOAApplication folder and select NISOAApplication.jws.

5. Click Open.

   The NISOAApplication.jws contains the NIDiscrepancyService project. The main components for this project are:

   - **NetworkIntegrityControlService.wsdl**: This is the Network Integrity Sample Web Services WSDL file.

   - **xds**: This folder contains Network Integrity Sample Web Service schema files.

   - **composite.xml**: This file describes the entire composite assembly of services, service components, references, and wires
In the project, composite.xml file is automatically created when the SOA project was created. In this application only service components (including Network Integrity Sample Web Service) are used.

- **NIBPELDiscurrenceProcess.bhel**: This file contains a list of variables and the main sequences in which Network Integrity Web Service calls to update the Network Integrity Discrepancies are defined. There are two parallel sequences named as **Sequence_1** and **Sequence_2** to update Attribute mismatch discrepancies for **nativeEMSServiceState** (go to step 6) and **physicalAddress** (go to step 8) respectively.

It is necessary that both client side artifacts (wsdl and schema) and server side artifacts are in sync and of same version.

6. To search for **nativeEMSServiceState** attribute mismatch discrepancies (**Sequence_1**), search for the following discrepancies:

   - **TYPE = ATTRIBUTE_VALUE_MISMATCH**
   - **ATTRIBUTEORRELATIONSHIPNAME = nativeEmsServiceState**
   - **STATUS = DISCREPANCY_OPENED**
   - **COMPARESOURCE = INVENTORY**
   - **REFERENCESOURCE = NETWORK using findDiscrepancy webservice operation.**

7. Loop over each discrepancy and submit to **updateDiscrepancy** if **COMPAREVALUE = 'IN_SERVICE'** and **REFERENCEVALUE = 'OUT_OF_SERVICE'** to update **OPERATION** as 'Correct in UIM' and **STATUS** as 'OPERATION_IDENTIFIED'.

8. To search for **physicalAddress attribute mismatch discrepancies**, search for the following discrepancies:

   - **TYPE = ATTRIBUTE_VALUE_MISMATCH**
   - **ATTRIBUTEORRELATIONSHIPNAME = physicalAddress**
   - **STATUS = DISCREPANCY_OPENED using findDiscrepancy webservice operation.**

9. Loop over each discrepancy and submit to **updateDiscrepancy** by setting **PRIORITY** to **High** and **DISCREPANCYOWNER** to given value.

10. Right-click **composite.xml** and select **Configure WS Policies** to add appropriate security client policy to the Network Integrity Web Service component.

11. Update NetworkIntegrityControlService.wsdl’s SOAP address location with Network Integrity Web Service URL. For example,

    `<soap:address location="https://<host_address>:<ssl_port>/NetworkIntegrityApp-NetworkIntegrityControlWebService-context-root/NetworkIntegrityControlServicePortType"/>

This should be done before building the SOA application or use deployment plan while deploying the SOA application to update the SOAP address location with the Network Integrity Web Service URL. This configuration is required for SOA application to communicate with Network Integrity Web Services.

**Starting and Stopping SOA Servers**

To start and stop SOA servers, use the following procedure:
1. To start the Admin server run to following command: `<domain>/startWeblogic.sh`

2. To start the SOA managed server, run the following command (here soa_server1 is name of SOA managed server): `<domain>/bin/startManagedServer.sh soa_server1`

3. To enter the WebLogic console, use:
   `http://<host_address>:7001/console`

4. To enter the Enterprise Manager console, use:
   `http://<host_address>:7001/em`

5. To enter SOA Infra, use:
   `http://<host_address>:8001/soa-infra`

6. Press Ctrl + C to stop the servers.

Building and Deploying the SOA Application

To build and deploy the SOA application, use the following procedure:

1. In Jdeveloper, go to Application Navigator then right-click `NIDiscrepancyService` project

2. Click `Make NIDiscrepancyService.jpr` in the menu to build the project. The project should build successfully without any compilation errors or warnings.

3. Start the Admin and SOA servers that are created as part SOA domain creation (see "Starting and Stopping SOA Servers" and "Creating WebLogic Domain with SOA Products").

4. Create a standalone server connection for the SOA server.

5. Right-click `NIDiscrepancyService` and select 'Deploy to Application server'.

6. The SOA suite provides an ant script to deploy and undeploy the SOA archive (SAR) file (deployable SOA application jar) in the BEA HOME. Use the following to deploy and undeploy the SAR file:

   - To deploy, use the following:
     ```
     ant -f <BEAHOME>/Oracle_SOA1/bin/ant-sca-deploy.xml
     -DserverURL=http://soa_server_host:soa_server_port
     -DsarLocation=<SOA archive file path>
     ```
     For example,
     ```
     ant -f /home/beahome/Oracle_SOA1/bin/ant-sca-deploy.xml
     -DserverURL=http://localhost:8001
     -DsarLocation=/home/example/beahome/mywork/NISOAApplication/NIDiscrepancyService/deploy/sca_NIDiscrepancyComposite_rev1.0.jar
     ```

   - To undeploy, use the following:
     ```
     ant -f <BEAHOME>/Oracle_SOA1/bin/ant-sca-deploy.xml undeploy
     -DserverURL=http://soa_server_host:soa_server_port
     -DcompositeName=<SOA composite name>
     -Drevision=<SOA composite version>
     ```
     For example,
     ```
     ant -f /home/beahome/Oracle_SOA1/bin/ant-sca-deploy.xml undeploy
     ```
Extensibility Tasks

-DserverURL=http://<localhost>:8001
-DcompositeName=NIDiscrepancyComposite
-Drevision=1.0

Testing Sample SOA application

To test a sample SOA application, use the following three tools:

- Testing Network Integrity SOA Application Using EM
- Testing Network Integrity SOA Application Using soa-infra
- Testing Network Integrity SOA Application Using SOAP UI Tool

Note: Oracle Enterprise Manager (EM) can also helpful in debugging and auditing of BPEL sequence exceptions.

Testing Network Integrity SOA Application Using EM

To test a sample SOA application with EM, use the following procedure:

1. Log on to the Enterprise manager as admin.
2. Expand the SOA folder to the deployed composite (NIDiscrepancyComposite).
3. Click Test to test composite.
4. Enter any value for the input argument for SOA Web Service.
5. Click Test Webservice. Wait for a response.
6. Click Launch Message Flow Trace to see detailed output.
7. Click NIBPELDiscrepancyProcess to view the Audit Trail, Flow, and so on.
8. Expand the payloads to see detailed input and output of each Web Service invoked.

Testing Network Integrity SOA Application Using soa-infra

To test a sample SOA application with soa-infra, use the following procedure:

1. Log on to soa-infra using the following URL:
   http://<host_address>:8001/soa-infra
2. Enter any input required for the test.
3. Click Invoke.

Testing Network Integrity SOA Application Using SOAP UI Tool

To test a sample SOA application with the Simple Object Access Protocol (SOAP) UI tool, use the following procedure:

1. Create a SOAP UI project at the following URL:
   http://<host_address>:8001/soa-infra/services/default/NIDiscrepancyComposite/nibpeldiscrepancyprocess_client_ep?WSDL
2. Enter any input required for the test.
3. Create a request run.
Network Integrity Localization Pack

This chapter provides information about the Localization Pack for Oracle Communications Network Integrity.

Localization Pack Overview

The following sections provide overview information about the localization pack:

- Localization Summary
- Localization Pack Overview
- Expected Outcomes

Localization Summary

The Network Integrity UI is a fully internationalized Web browser-based application. English is the default interface language. A System Integrator can localize the UI into any other languages, by building a Localization Pack.

Localization Pack Summary

The Network Integrity UI makes use of the full depth of i18n support provided by the Application Development Framework (ADF) stack. The application UI is fully internationalized by making use of XML Localization Interchange File Format (XLIFF) files to keep all display strings separate from other code artifacts. Various parts of the ADF stack (ADF Faces, ADF Model, and ADF Data Control) are also built with full i18n support. A Localization Pack is a collection of XLIFF files and other property files, which together localizes the UI to another language. A Localization Pack can be installed into the Network Integrity application, in the same manner as a Cartridge Pack installation.

Expected Outcomes

The user can successfully build a Localization Pack.

Building the Localization Pack

Use the following procedure to build a Localization Pack:

1. Download localization.iar from the Localization Pack in the Oracle Communications Network Integrity documentation media pack on the Oracle software delivery Web site:

   https://edelivery.oracle.com
2. Extract META-INF/manifest.xml to a temporary location.

3. Open manifest.xml and edit the value of the name tag:

   `<name>Locale_Name (Locale_Code)</name>

Where Locale_Name is the locale of the localization pack that you want to deploy in Network Integrity; for example, German, and where Locale_Code is the code of the locale of that localization pack; for example, de, as in the following example:

   `<name>German (de)</name>

For information on supported locales and valid locale codes, see the following Java guide:

   http://download.oracle.com/javase/1.4.2/docs/guide/intl/locale.doc.html

4. Save manifest.xml and return it to localization.iar/META-INF.

5. Extract META-INF/cartridge.xml to a temporary location.

6. Open cartridge.xml and edit the values of the name and languageCode tags:

   `<localizations>
     <localization>
       <name>Locale_Name</name>
       <languageCode>Locale_Code</languageCode>
     </localization>
   </localizations>

Where Locale_Name is the locale of the localization pack that you want to deploy in Network Integrity; for example, German, and where Locale_Code is the code of the locale of that localization pack; for example, de, as in the following example:

   `<localizations>
     <localization>
       <name>German</name>
       <languageCode>de</languageCode>
     </localization>
   </localizations>

7. Save cartridge.xml and return it to localization.iar/META-INF.

8. Extract localization.iar/localization.jar to a temporary location.

9. Extract localization.jar/oracle to a temporary location.

10. Edit all the XLF files found in localization.jar/oracle or any of its nested folders:

    a. Edit the name of each XLF file to add an underscore and the locale code before the file extension, as shown in the following example:

       DisAddressMsgBundle_de.xlf
b. Open each XLF file and edit the **file** tag so that the **source-language** attribute is set to the locale code, as in the following example:

```xml
<file source-language="de" original="oracle.communications.inVENTORY.api.entity.PhysicalPortMsgBundle" datatype="xml">
```

**Note:** The **source-language** attribute for compound locale codes, such as ar-QA, should be set to the first two characters only, as in the following example:

```xml
<file source-language="ar" original="oracle.communications.inVENTORY.api.entity.PhysicalPortMsgBundle" datatype="xml">
```

b. Open each XLF file and edit the file tag so that the **source-language** attribute is set to the locale code, as in the following example:

```xml
<file source-language="de" original="oracle.communications.inVENTORY.api.entity.PhysicalPortMsgBundle" datatype="xml">
```

**Note:** The **source-language** attribute for compound locale codes, such as ar-QA, should be set to the first two characters only, as in the following example:

```xml
<file source-language="ar" original="oracle.communications.inVENTORY.api.entity.PhysicalPortMsgBundle" datatype="xml">
```

c. Open each XLF file, locate each **trans-unit** tag and edit its child **source** tag with the translated value for the desired localization.

11. Edit all the PROPERTIES files found in localization.jar/oracle or any of its nested folders:

a. Edit the name of each PROPERTIES file to add an underscore and the locale code before the file extension, as shown in the following example:

```properties
IntegrityUIBundle_de.properties
```

**Note:** Compound locale codes, such as ar-QA, should be added to the XLF file name with an underscore in the place of the hyphen, as in the following example:

```properties
IntegrityUIBundle_ar_QA.properties
```

b. Open each PROPERTIES file and edit the value for each key with the translated value for the desired localization. For example, edit the **INTEGRITY_MANAGE_SCAN_CONFIG** key, as in the following example:

```properties
INTEGRITY_MANAGE_SCAN_CONFIG=new_value
```

Where **new_value** is the translated value for the key for the desired localization.

c. (Optional) To enter extended character values (such as Chinese characters), you must use Unicode Escapes (only one character is allowed per escape sequence). Save each PROPERTIES file with UTF-8 encoding, then convert each PROPERTIES file to Unicode Escapes using the native2ascii tool provided with your JDK by entering the following command:

```
native2ascii -encoding UTF-8 input_file_name output_file_name
```

Where **input_file_name** is the name of the PROPERTIES file being converted, and where **output_file_name** is the name of the converted file.

See the partial sample Chinese localization included in the Localization Pack for an example.
12. Return all XLF and PROPERTIES files to `localization.jar`.

13. Return `localization.jar` to `localization.iar`.


15. (Optional) To localize link names in the Link panel in the Network Integrity UI, you must edit the MBean with the translated values for the desired localization. See `Network Integrity System Administrator’s Guide` for more information about viewing and editing the MBean.

16. (Optional) To localize cartridge-specific UI parameters:
   a. Extract the contents from your custom cartridge IAR file to a temporary location.
   b. Extract the cartridge MAR file to a temporary location.
   c. Extract the `/resourcebundles/xliffBundles/IntegrityOverrideBundle.xlf` file to a temporary location.
   d. Edit `IntegrityOverrideBundle.xlf`:
      - Open each XLF file and edit the `file` tag so that the `source-language` attribute is set to the locale code, as in the following example:
        ```xml
        <file source-language="de" original="oracle.communications.inventories.api.entity.PhysicalPortMsgBundle" datatype="xml">
        ```
      - Open each XLF file, locate each `trans-unit` tag and edit its child `source` tag with the translated value for the desired localization.
   e. Return `IntegrityOverrideBundle.xlf` to the cartridge MAR file.
   f. Return the cartridge MAR file to the cartridge IAR file.
   g. Deploy your custom cartridge.

Testing the Browser Local

When running the Network Integrity UI, the user chooses the appropriate language from the web browser. This is usually done using the Character or Text Encoding menu of the browser, or from a Language preference setting. The UI displays the selected language only when the corresponding Localization Pack is installed. Otherwise, the UI displays the default English language.

There may be parts of the UI that are supplied by third parties, which are not fully internationalized. Those parts always display in English.

Note: The `source-language` attribute for compound locale codes, such as ar-QA, should be set to the first two characters only, as in the following example:

```xml
<file source-language="ar" original="oracle.communications.inventories.api.entity.PhysicalPortMsgBundle" datatype="xml">
```