Oracle® Communications Network Integrity

Incremental TMF814 Discovery Cartridge Guide

Release 7.3.2

E66041-01

May 2016

ORACLE®

Oracle Communications Network Integrity Incremental TMF814 Discovery Cartridge Guide, Release 7.3.2

E66041-01

Contents

5 About Cartridge Modeling

6 About Design Studio Construction

7 About Design Studio Extension

8 NMS Notification Listener Sample Application Reference

Preface

This guide explains the functionality and design of the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge.

Audience

This guide is intended for Network Integrity administrators, developers, and integrators.

This guide assumes that you are familiar with the following documents:

- *Network Integrity Developer's Guide*: for basic understanding of cartridges
- *Network Integrity Installation Guide*: for information about deploying and undeploying cartridges
- *Network Integrity CORBA Cartridge Guide*: for an understanding of the functionality and design of the Network Integrity Cartridge for CORBA (CORBA cartridge)
- *Network Integrity Optical TMF814 CORBA Cartridge Guide*: for an understanding of the functionality and design of the Optical TMF814 CORBA cartridge

This guide assumes that your are familiar with the following concepts:

- TMF814 standards and terminology
- Common object request broker architecture (CORBA) standards and terminology
- Oracle Communications Design Studio
- Oracle Communications Information Model

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 that have purchased support 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.

Document Revision History

The following table lists the revision history for this guide:

Version	Date	Description
E66041-01	May 2016	Initial release.

1

Overview

This chapter describes the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge.

About the Incremental TMF814 Discovery Cartridge

The Incremental TMF814 Discovery cartridge is an extension to the existing Network Integrity Optical TMF814 CORBA cartridge. See *Network Integrity Optical TMF814 CORBA Cartridge Guide* for more information.

The Incremental TMF814 Discovery cartridge is dependent on the Optical TMF814 CORBA cartridge. You must import the Optical TMF814 CORBA cartridge before importing the Incremental TMF814 Discovery cartridge in Oracle Communications Design Studio.

The Incremental TMF814 Discovery cartridge can perform a full scan (same as the Optical TMF814 CORBA cartridge) or an incremental discovery scan. In an incremental scan, the Incremental TMF814 Discovery cartridge discovers only the network elements, such as managed elements (MEs), topological links (TLs), and subnetwork connections (SNCs), that have changed in the network since the last discovery scan; thus avoiding the need to discover the entire network. This cartridge allows you to make faster inventory updates by reducing the time taken for network discovery and reconciliation.

The incremental discovery scan retrieves information using an iterative query approach. Address-based discovery retrieves the complete information in a single communication; as a result, the incremental discovery scan is not applicable in such scenarios.

The Incremental TMF814 Discovery cartridge solution explained in this guide discovers and models physical and logical entities for the following vendors: Huawei and ECI.

The Incremental TMF814 Discovery cartridge consists of the following components:

- **NMS Notification Listener Sample Application:** This sample application is connected to multiple element management system (EMS) servers or network management system (NMS) servers to listen to event channels and capture the NMS notifications that are generated when network elements are created, updated, or deleted. See "NMS Notification Listener Sample Application" for more information.
- **NetworkIntegritySDK cartridge:** The NetworkIntegritySDK cartridge is enhanced to support incremental discovery. See *Network Integrity Developer's Guide* for more information.

The Incremental TMF814 Discovery cartridge uses the TMF814 CORBA interface as a discovery protocol to connect to the EMS or NMS servers and retrieve information about network elements from that system.

The Incremental TMF814 Discovery cartridge discovers network elements in the network using a TMF814 common object request broker architecture (CORBA) interface. This cartridge provides discovery actions capable of discovering both physical (equipment) and logical (interface) hierarchy details of MEs.

The Incremental TMF814 Discovery cartridge can be used to discover the following network systems:

- Synchronous Optical Networking (SONET)
- Synchronous Digital Hierarchy (SDH)
- Dense Wavelength Division Multiplexing (DWDM)
- Asynchronous Transfer Mode (ATM)
- Ethernet

NMS Notification Listener Sample Application

The NMS Notification Listener Sample Application is connected to multiple EMS and NMS servers to listen to event channels and capture the NMS notifications that are generated when network elements are created, updated, or deleted. The NMS Notification Listener Sample Application is a notification listener based on version 1.0.1 of the Object Management Group (OMG) Notification Service specification. The NMS Notification Listener Sample Application supports all the EMS and NMS notification services that are compliant with version 1.0.1 of the OMG Notification Service specification.

You can extend the NMS Notification Listener Sample Application to support vendor-specific EMS and NMS notification services that you may have implemented.

For information about the OMG Notification Service specification, see the OMG Web site at:

http://www.omg.org/spec/NOT/

The Notification Service specification supports event types defined by the OMG Event Service. The OMG Event Service does the following:

- Supports asynchronous exchange of notification events between clients.
- Introduces event channels that communicate notification events between suppliers and consumers by issuing standard CORBA requests.
- Defines two roles for objects:
  - **Suppliers:** Event Service clients that produce notification events
  - **Consumers:** Event Service clients that consume and process notification events

The OMG Notification Service specification defines various types of suppliers and consumers. The NMS Notification Listener Sample Application implements the StructuredPushConsumer interface that listens for the notification events of the NMS server and persists the relevant information (such as native EMS names and last modified date and time of MEs, SNCs, and TLs) in standard file format. These text files act as an input to the Discover Incremental TMF814 scan, which does the following:

- Identifies and retrieves only those network elements (from NMS Server) that have changed in the network since the last discovery scan and displays them in the scan results
- Copies the unmodified network elements from the previous successful scan and displays them in the scan results

The NMS Notification Listener Sample Application uses OMG CORBA Notification Service standard APIs to connect to the NMS. You must install and set up the Notification Listener Sample Application to receive notification events from the NMS and to communicate the events to Network Integrity. The Notification Listener Sample Application is packaged separately and is available along with the NetworkIntegritySDK cartridge project. You must change the Notification Listener Sample Application configuration using a Java editor to connect to the NMS server and receive its notification events.

See "NMS Notification Listener Sample Application Reference" for more information about the example code for the NMS Notification Listener Sample Application.

About Cartridge Dependencies

This section provides information about dependencies that the Incremental TMF814 Discovery cartridge has on other entities.

Run-Time Dependencies

There are no run-time dependencies for this cartridge.

Design Studio Dependencies

To load the Incremental TMF814 Discovery cartridge into Oracle Communications Design Studio, the following cartridge projects must be installed:

- TMF814Discovery_Cartridge
- Abstract_CORBA_Cartridge
- NetworkIntegritySDK
- Address_Handlers
- ora_uim_model

Opening the Cartridge Files in Design Studio

To review and extend the Incremental TMF814 Discovery cartridge, download the Incremental TMF814 Discovery cartridge ZIP file from the Oracle software delivery Web site:

https://edelivery.oracle.com/

The Incremental TMF814 Discovery cartridge ZIP file has the following structure:

- **\UIM_Cartridge_Projects\TMF814_Model**
- **\Network_Integrity_Cartridge_Projects\Incr_TMF814_Disc_Cartridge**

The **Incr_TMF814_Disc_Cartridge** project contains the extendable Design Studio files.

You must open the files in Design Studio before you can review and extend the cartridge.

See *Network Integrity Concepts* for guidelines and best practices for extending cartridges. See *Network Integrity Developer's Guide* for information about opening files in Design Studio.

Building and Deploying the Cartridge

See the Design Studio Help for information about building and deploying cartridges.

2

About the Cartridge Components

This chapter provides information about the components that make up the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge.

The Incremental TMF814 Discovery cartridge is made up of the following components:

- Abstract Incremental Discovery Action
- Discover Incremental TMF814 Action

See "About Design Studio Construction" for information about how the actions are built.

The Incremental TMF814 Discovery cartridge supports a recording mode for recording TMF814 data. See "About Recording Mode" for more information.

Abstract Incremental Discovery Action

The Abstract Incremental Discovery action validates the scan parameters and verifies whether incremental scan is enabled. This action outputs a parameter reference to the last successful scan. If no previous scans exist, the complete scan is performed.

The Abstract Incremental Discovery action contains the following processor:

- Incremental Discovery Initializer

Incremental Discovery Initializer

The Incremental Discovery Initializer processor performs the validation of scan parameters and sets the scan to run incrementally.

Discover Incremental TMF814 Action

The Discover Incremental TMF814 action, which extends the Discover Abstract TMF814 and Abstract Incremental Discovery actions, discovers only the network elements, such as managed elements (MEs), topological links (TLs), and subnetwork connections (SNCs), that have changed in the network since the last discovery scan.

This Discover Incremental TMF814 action inherits all the processors from the following actions:

- The Discover Abstract TMF814 action

  For information about the inherited processors in this action, see *Network Integrity Optical TMF814 CORBA Cartridge Guide*.

- The Abstract Incremental Discovery action

For information about the inherited processors in this action, see "Abstract Incremental Discovery Action".

The Discover Incremental TMF814 action contains the following processors run in the following order:

1. TMF814 Property Initializer (inherited)
2. TMF814 Session Manager (inherited)
3. TMF814 Device Recorder Initializer (inherited)
4. Incremental Discovery Initializer (Inherited)
5. TMF814 ME Collector (inherited)
6. TMF814 Updated ME Discoverer
7. TMF814 Device Modeler (inherited)
8. TMF814 Equipment Collector (inherited)
9. TMF814 Equipment Modeler (inherited)
10. TMF814 PTP Collector (inherited)
11. TMF814 PTP Modeler (inherited)
12. TMF814 CTP Discoverer for PTP (inherited)
13. TMF814 FTP Collector (inherited)
14. TMF814 FTP Modeler (inherited)
15. TMF814 CTP Discoverer for FTP (inherited)
16. TMF814 Device Persister (inherited)
17. TMF814 Device Recorder Persister (inherited)
18. TMF814 Cross-Connect Discoverer (inherited)
19. TMF814 SNC Discoverer
20. TMF814 SNC CC Discoverer
21. TMF814 Topological Link Collector (inherited)
22. TMF814 Updated TL Discoverer
23. TMF814 Topological Link Modeler (inherited)
24. TMF814 Pipe Persister (inherited)

Figure 2–1 illustrates the processor workflow of the Discover Incremental TMF814 action.

Figure 2–1 Discover Incremental TMF814 Action Processors

TMF814 Updated ME Discoverer

This processor accepts the complete list of discovered devices, identifies the modified devices, and copies the previous successful scan results for unmodified devices.

TMF814 SNC Discoverer

This processor discovers the configured subnetwork connections under the network management system (NMS).

TMF814 SNC CC Discoverer

This processor discovers the cross-connects under the specified subnetwork, models the cross-connects, and persists the modeled cross-connects.

TMF814 Updated TL Discoverer

This processor does the following:

- Queries all the TL names
- Checks the updated TL names from the custom handler and identifies the TLs yet to be discovered
- Discovers the specific TLs using CORBA API
- Copies the unmodified TL entities from the previous successful scan

About Recording Mode

You can configure the Incremental TMF814 Discovery cartridge to record all discovered MEs, topological links, and cross-connects. When recording mode is enabled, the recorded files (*ME_Name***.me** for MEs, *EMS_Name***.ems** for topological

links, and *EMS_Name***.cc** for cross-connects) are saved to the *Domain_Home*/**corbaData/***Scan_Name/EMS_Name* directory, where:

- *ME_Name* is the name of the managed element.
- *EMS_Name* is the name of the EMS.
- *Domain_Home* is the directory containing the configuration for the domain into which Network Integrity is installed.
- *Scan_Name* is the name of the scan.

See "Enabling and Disabling Recording Mode" for instructions on how to enable the recording mode.

Enabling and Disabling Recording Mode

The recording mode can be enabled or disabled by an administrator without needing to restart the server or application. The recording mode you set takes effect immediately.

To enable or disable recording mode:

1. Open the *Domain_Home*/**config/corbaConfig/tmf814.properties** file.
2. Do one of the following:
   - To disable recording mode, locate the **MODE** entry and set its value to **NORMAL**.
   - To enable recording mode, do the following:
     - Locate the **MODE** entry and set its value to **RECORD**.
     - Set the **CHUNK SIZE** entry to the number of cross-connects written to *EMS_Name***.cc** at a time.

3

Using the Cartridge

This chapter explains how to use the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge.

Creating a Discover TMF814 Scan

The Incremental TMF814 Discovery cartridge allows you to create a Discover TMF814 scan.

To create an Incremental TMF814 Discovery scan:

1. Create a scan, as explained in the Network Integrity Help.
2. On the **General** tab, from the **Scan Action** list, select **Discover Incremental TMF814** to discover only the modified network elements (MEs, SNCs, and TLs) from the network management system.

   The **Scan Type** field displays `Discovery`.
3. Under the **Scan Action Parameters** area, from the **Select Parameter Group** list, select the following:
   - Select **TMF814Parameters** and enter the following TMF14 scan action parameters:
     - In the **Username** field, enter the username for the target element or network management system (EMS or NMS).
     - In the **Password** field, enter the password for the target EMS or NMS.
     - In the **EMS Naming Service** field, enter the EMS session factory CORBA object name.
     - From the **EMS Naming Service Format** list, specify whether the EMS session factory CORBA object name uses the **Plain**, or the **Stringified** format.
     - From the **Collect Equipment** list, specify whether you want to collect equipment holder objects.
     - From the **Collect Termination Points** list, specify the type of termination points (TPs) you want to collect. To not collect any TPs, select **None**.
     - From the **Collect Connection TP** list, specify the type of connection TPs you want to collect. To not collect any connection TPs, select **None**.
     - (Optional) To set the number of equipment objects to retrieve with each EMS call, enter a value in the **Equipment Fetch Size** field. Leave this field blank to retrieve all equipment objects in a single EMS call.

- (Optional) To set the number of TPs to retrieve with each EMS call, enter a value in the **Termination Point Fetch Size** field. Leave this field blank to retrieve all TPs in a single EMS call.
- (Optional) To set the number of hierarchical child levels to which contained TPs are collected, enter a value in the **Contained TP Collection Depth** field. Leave this field blank to retrieve all contained TPs.
- (Optional) To pass custom object request broker (ORB) properties to the Discover Abstract TMF814 action, enter name value pairs in the **ORB Properties** field, separated by a semicolon, as in the following example:

  ```
  Property_1=value_1;Property_2=value_2;Property_n=value_n
  ```

- (Optional) To pass custom ORB arguments to the Discover Abstract TMF814 action, enter name value pairs in the **ORB Arguments** field, separated by a semicolon, as in the following example:

  ```
  Argument_1=value_1;Argument_2=value_2;Argument_n=value_n
  ```

- (Optional) To filter the discovered managed elements (MEs) by name, enter a name in the **Managed Element Name(s)** field and set the **Managed Element Name Qualifier** list.
- (Optional) To filter the discovered network elements (NEs) by name, enter a name in the **Network Element Name(s)** field and set the **Network Element Name Qualifier** list.
- In the **Cross Connect Collection Type** field, specify how cross-connect objects are collected. Irrespective of whether you select **SNC** or **ME list**, only the **SNC** option is considered for fetching cross-connects. To not collect any cross-connect objects, select **None**.
- In the **Topological Link Collection Type** field, specify how topological links are collected. To not collect any topological links, select **None**.

See Table 6–3, " TMF814 Scan Parameters Design Studio Construction" for more information.

- (Optional) Select **AutoResolutionParameter** and then select the **Auto Resolve Discrepancies** check box to initiate automatic discrepancy resolution at the end of the scan. Automatic discrepancy resolution works only if the **Detect Discrepancies** option is also selected. See *Network Integrity Developer's Guide* for more information.
- (Optional) Select **IncrementalScanParameter** and then select the **Incremental Scan** check box.

4. On the **Scope** tab, do one of the following:
   - Enter the EMS CORBA Loc URL.
   - Import the IOR file.
   - Enter the content of the IOR file.

   Note: All entries on the **Scope** tab must be unique. All entries are validated against the CorbaURLAddressHandler address handler.

5. On the **Schedule** tab, define a schedule for the scan.

See the Design Studio Modeling Network Integrity Help for more information on defining a scan schedule.

6. Make any other required configurations.
7. Click **Save and Close**.

Extending the Incremental Discovery Cartridge to Support Different Protocols

You may be required to extend the existing Incremental Discovery cartridge (sample NMS listener application) to support different protocols used by vendor-specific EMS and NMS notification services that you may have implemented.

To extend the Incremental Discovery Cartridge to support different protocols used by an EMS or NMS:

1. Continuously monitor the changes made to the network devices by listening to the notifications of the EMS or NMS.
2. Enhance the discovery cartridge by adding a new discovery processor that does the following:
   - Checks the text file generated by the notification listener for the list of devices that are modified after the previous scan by going through the device tracking file
   - Fetches only the modified devices from the network
   - Copies the unmodified devices from the previous scan
   - Removes the unmodified devices from the list of devices to be discovered

4

About Collected Data

This chapter explains how the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge treats collected data.

About Collected Data

The Incremental TMF814 Discovery cartridge uses a standard TMF814 common object request broker architecture (CORBA) interface, which models network elements using the Multi Technology Network Management (MTNM) standard.

Table 4–1 lists MTNM objects and corresponding TMF814 IDL API class definitions.

Table 4–1 MTNM IDL Class Definitions

MTNM Object Name	TMF814 IDL API Class Definition
Managed Element (ME)	ManagedElement_T
Equipment Holder (Rack)	EquipmentHolder_T
Equipment Holder (Shelf)	EquipmentHolder_T
Equipment Holder (Slot)	EquipmentHolder_T
Equipment Holder (Sub Slot)	EquipmentHolder_T
Equipment (Card)	Equipment_T
Physical Termination Point (PTP)	TerminationPoint_T
Floating Termination Point (FTP)	TerminationPoint_T
Connection Termination Point (CTP)	TerminationPoint_T
Cross-connect	CrossConnect_T
Topological Link	TopologicalLink_T
LayeredParameters	LayeredParameters_T

A CTP can have a child CTP with any number of nesting levels. LayeredParameters are not top-level MTNM objects. They are a property of a termination point (TP).

Multi Technology Network Management Hierarchy

The following example demonstrates the MTNM hierarchy:

```
Managed Element
  Equipment Holder(rack 1)
    Equipment Holder (shelf 1)
      Equipment Holder (sub shelf 1)
```

```
Equipment Holder (slot 1)
  Equipment Holder(sub slot 1)
    Equipment(card 1)
      Termination Point (PTP){0…*}
        Termination Point (CTP){0…*}
  Equipment Holder(sub slot 2)
    Equipment(card 2)
      Termination Point (PTP){0…*}
        Termination Point (CTP){0…*}
          Termination Point (CTP){0…*}
```

The following tables describe the properties of each MTNM object collected by the Incremental TMF814 Discovery cartridge:

- Table 4–2, " Managed Element Properties"
- Table 4–3, " Equipment Properties"
- Table 4–4, " Equipment Holder Properties"
- Table 4–5, " PTP, FTP, and CTP Properties"
- Table 4–6, " Cross-Connect Properties"
- Table 4–7, " Topological Link Properties"

Table 4–2 Managed Element Properties

Property Name	Description
name	The name of the managed element (ME) that is assigned by the element management system (EMS) upon creation.
userLabel	Identifies the label assigned to the ME by the operator.
nativeEMSName	Indicates how the ME is referred to on EMS displays.
owner	Provided by the network management system (NMS).
location	Indicates the geographical location of the ME.
version	The active software version of the ME.
productName	Identifies the ME product or type name.
communicationState	Indicates the viability of EMS-ME messaging. Possible values are CS_AVAILABLE, CS_ UNAVAILABLE.
emsInSyncState	Indicates if the EMS is able to keep the current EMS data synchronized with the current ME data and generate all appropriate notifications. The EMS sets this attribute to FALSE to indicate that it requires re-synchronization with ME data and that it is not able to generate the appropriate notifications while doing so.
supportedRates	This attribute is a list of potential cross-connection rates at which it is possible to have cross-connections within the ME.
additionalInfo	Represents a list of attributes that are EMS and NMS implementation specific. This field is common to all MTNM-managed objects. This field consists of a list of name and value pairs that call additional information, which allow the EMS or NMS to give additional information that is not explicitly modeled using the MTNM interface. Some parameter names and values may be predefined.

Table 4–3 Equipment Properties

Property Name	Description
name	The name of the Equipment that is assigned by the EMS upon creation.
nativeEMSName	Indicates how the Equipment is referred to on EMS displays.
userLabel	A label assigned to the Equipment by the operator.
owner	Provided by the NMS.
alarmReportingIndicator	Indicates whether alarm reporting for this instance is active.
expectedEquipmentObjectType	Defines the type of expected Equipment. Leave empty if there is no expected Equipment. Example value: MBP_300.
installedEquipmentObjectType	Defines the type of installed Equipment. Leave empty if there is no installed Equipment.
installedPartNumber	Indicates the part number of the installed Equipment.
installedSerialNumber	Indicates the serial number of the installed Equipment.
installedVersion	Indicates the firmware version of the installed Equipment.
serviceState	Indicates the current state of the Equipment. Possible values are IN_SERVICE, OUT_OF_SERVICE, OUT_OF_SERVICE_BY_MAINTENANCE, SERV_NA.
additionalInfo	Represents a list of attributes that are EMS and NMS implementation specific. This field is common to all MTNM-managed objects. This field consists of a list of name and value pairs that call additional information and allow EMSs or NMSs to give additional information that is not explicitly modeled at the MTNM interface. Some parameter names and values may be predefined.

Table 4–4 Equipment Holder Properties

Property Name	Description
name	Equipment Holder unique name. The EMS is responsible for the uniqueness of the name within the context of the ME.
nativeEMSName	Indicates how the Equipment Holder is referred to on EMS displays.
userLabel	Provided by the NMS.
owner	Provided by the NMS.
alarmReportingIndicator	Indicates whether alarm reporting is active for the instance.
holderType	Indicates the type of Equipment Holder. Valid values are: rack, shelf, sub_shelf, slot, sub_slot.
holderState	Indicates the state of the Equipment Holder directly contained equipment. Possible values are: EMPTY (0), INSTALLED_AND_EXPECTED (1), EXPECTED_AND_NOT_INSTALLED (2), INSTALLED_AND_NOT_ EXPECTED (3), MISMATCH_OF_INSTALLED_AND_EXPECTED (4), UNAVAILABLE (5), UNKNOWN (6).
expectedOrInstalledEquipment	The Equipment object expected or installed in the Equipment Holder, if any. A value of NULL indicates that the Equipment Holder is empty or that it contains only other Equipment Holders.
acceptableEquipmentTypeList	Represents the types of Equipment objects that can be directly supported by the Equipment Holder.
additionalInfo	Represents a list of attributes that are EMS and NMS implementation specific. This field is common to all MTNM-managed objects. This field consists of a list of name and value pairs that call additional information, which allow the EMS or NMS to give additional information that is not explicitly modeled at the MTNM interface. Some parameter names and values may be predefined.

Table 4–5 PTP, FTP, and CTP Properties

Property Name	Description
name	Indicates the assigned TP name when created by the EMS. The EMS is responsible for guaranteeing the uniqueness of the name within the context of the ME. The naming for CTPs, PTPs, and FTPs is deterministic.
nativeEMSName	Indicates how the TP is referred to on EMS displays.
userLabel	The user label of the TP is set with NMS data (typically the end-to-end trail data).
owner	Indicates the ownership of the TP so that adminstrativeState can be managed.
direction	Indicates the direction of the TP. Possible values are: D_NA (0), D_BIDIRECTIONAL (1), D_SOURCE (2), D_SINK (3).
tpProtectionAssociation	Indicates the associated TP indication. The NMS is responsible for running the multiLayerSubnetwork::MultiLayerSubnetworkMgr_I::getAssociatedTP() service to obtain any related TP.
edgePoint	Indicates if the TP is an edge point of one or more subnetworks.
ingressTransmissionDescriptorName	Indicates whether a CTP references an ingress (incoming) Traffic Descriptor or Transmission Descriptor.
egressTransmissionDescriptorName	Indicates whether a CTP references an egress (outgoing) Traffic Descriptor or Transmission Descriptor.
connectionState	Indicates the connection state of the source. A value of TPCS_BI_CONNECTED indicates that the source is connected to one entity and the sink is connected to the other. Possible values are: TPCS_NA, TPCS_SOURCE_CONNECTED, TPCS_SINK_CONNECTED, TPCS_BI_CONNECTED, TPCS_NOT_CONNECTED.
tpMappingMode	Indicates and controls the connection of the named connection point at a specified LayerRate to the dedicated G.805 TCP and associated G.805 Termination Function at the same LayerRate within the CTP or FTP. Possible values are: TM_NA (0), TM_NEITHER_TERMINATED_NOR_AVAILABLE_FOR_MAPPING (1), TM_TERMINATED_AND_AVAILABLE_FOR_MAPPING (2).
type	Possible value are: TPT_PTP (0), TPT_CTP (1), TPT_TPPool (2).
transmissionParams	A list of transmission parameters that can be set or retrieved on the TP at a specified layer. This attribute must contain the complete set of layer rates represented by a PTP, CTP, or FTP, even if they have no parameters associated with them. The Layer Rates are listed in the order of their client-server relationship.
additionalInfo	Represents a list of attributes that are EMS and NMS implementation specific. This field is common to all MTNM-managed objects. This field consists of a list of name and value pairs that call additional information, which allows the EMS or NMS to give additional information that is not explicitly modeled at the MTNM interface. Some parameter names and values may be predefined.

Table 4–6 Cross-Connect Properties

Property Name	Description
active	Indicates if the cross-connect is active in the ME.
ccType	Indicates the cross-connect type. Possible values are: ST_SIMPLE, ST_ADD_DROP_A, ST_ADD_DROP_Z, ST_INTERCONNECT, ST_DOUBLE_INTERCONNECT, ST_DOUBLE_ADD_DROP, ST_OPEN_ADD_DROP, ST_EXPLICIT.
direction	Directionality of the cross connection. Possible values are: CD_UNI, CD_BI.

Table 4–6 (Cont.) Cross-Connect Properties

Property Name	Description
aEndNameList	Names of CTPs, FTPs, and group termination points (GTPs) at the aEnd of the cross-connect.
zEndNameList	Names of CTPs, FTPs, and GTPs at the zEnd of the cross-connect.
additionalInfo	Represents a list of name value pairs that allow EMSs or NMSs to give additional information that is not explicitly modeled at the MTNM interface. Some parameter names and values may be predefined. Some predefined parameter names may include: ConnectionId, Fixed, RouteActualState, RouteAdminState, RouteExclusive, RouteId, RouteIntended, RouteInUseBy.

Table 4–7 Topological Link Properties

Property Name	Description
name	Indicates the name of the Topological Link, assigned by the EMS upon creation.
userLabel	Indicates the topological link user label (end-to-end trail data) in NMS data.
nativeEMSName	Indicates how the topological link is referred to on EMS displays.
owner	Provided by the NMS.
direction	Indicates the direction of the topological link. A topological link can be unidirectional even if both its ends are bidirectional TPs. Possible values are CD_UNI (unidirectional) and CD_BI (bidirectional).
rate	Indicates the layer rate (bandwidth) of the topological link.
aEndTP	Indicates the name of the aEnd for the PTP, CTP, or FTP.
zEndTP	Indicates the name of the zEnd for the PTP, CTP, or FTP.
additionalInfo	Represents a list of name/value pairs that allow EMSs or NMSs to give additional information that is not explicitly modeled at the MTNM interface. Some parameter names and values may be predefined. Some predefined parameter names may include: AlarmReporting, AllocatedNumber, ASAPpointer, FragmentServerLayer, NetworkAccessDomain.

Layer Parameters

The Incremental TMF814 Discovery cartridge collects layer parameters for TPs. In the MTNM model, these layer parameters are encapsulated by TPs as transmission parameters. For details on layered parameters see the TM Forum documentation.

TMF814 APIs

This section describes the APIs used by the Incremental TMF814 Discovery cartridge to collect data.

CORBA APIs

Table 4–8 lists the APIs used by the Incremental TMF814 Discovery cartridge.

Table 4–8 TMF814 Managed Element and Equipment CORBA APIs

API	Used Operations
org.tmforum.mtnm.emsSessionFactory.EmsSessionFactory_I	▪ getEmsSession(): used to obtain the EmsSession objects.
org.tmforum.mtnm.emsSession.EmsSession_I	▪ getManager(): used to obtain managers. ▪ endSSession(): used to close the EMS session.
org.tmforum.mtnm.managedElementManager.ManagedElementMgr_I	▪ getAllFTPs(): used to obtain all FTPs, but does not obtain PTPs. ▪ getAllPTPs(): used to obtain all PTPs. ▪ getContainedInUseTPs(): used to obtain all contained in-use TPs. ▪ getContainedPotentialInUseTPs(): used to obtain all contained potential CTPs for a given TP.
org.tmforum.mtnm.nmsSession.NmsSession_I	▪ EmsSessionFactory_I.getEmsSession: required dummy session you must provide to get the actual EMS session.
org.tmforum.mtnm.equipment.EquipmentInventoryMgr_I	▪ getAllEquipment(): used to obtain all Equipment. ▪ getAllSupportedPTP(): used to obtain all the PTPs for a given Equipment.

APIs for Cross-Connect Collection

Table 4–9 lists the APIs used for cross-connect collection.

Table 4–9 TMF814 Cross-Connect Collection APIs

API	Used Operations
managedElementManager.ManagedElementMgr_I	▪ getAllCrossConnections(MEName, layerRate, how_many, CClist, CCIter)
multiLayerSubnetwork.MultiLayerSubnetworkMgr_I	▪ getAllTopLevelSubnetworks(how_many, holder, iter) ▪ getAllSubnetworkConnections(SN_Name, layerRateList, how_many, holder, iter) ▪ getRoute(SNC_Name, includeHigherOrderCCs, route)

APIs for Topological Link Collection

There are two levels of Topological Links that can be retrieved using two different APIs. Table 4–10 lists the APIs used for topological link collection.

Table 4–10 TMF814 Topological Link Collection APIs

API	Used Operations
emsMgr.EMSMgr_I	▪ getAllTopLevelTopologicalLinks(how_many, topoList, topoIt)
multiLayerSubnetwork.MultiLayerSubnetworkMgr_I	▪ getAllTopologicalLinks(SN_Name, how_many, topoList, topoIterator)

The ElementManagementSystemMgr (EMSMgr) API is used when the entire network is treated as a subnetwork. The MultiLayerSubnetworkMgr (MLSN) API is used when each ME is treated as a subnetwork.

Handling Vendor Variations

This section explains how the Incremental TMF814 Discovery cartridge handles certain data collected from some vendors.

FTP Collection API Variations

The ManagedElementMgr_I.getAllFTP() operation, from MTNM version 3.0, is the preferred API for getting all FTPs of an ME. For the vendors and devices that do not support MTNM version 3.0, the getAllPTP() operation is used. The getAllPTP() operation returns both PTPs and FTPs. While modeling FTPs, PTPs are filtered out.

Cross-Connect Collection API Variation

Cross-connects are collected using different APIs depending on the vendor. Use the crossConnectCollectionType parameter to specify the collection method, based on vendor device specifications. See "APIs for Cross-Connect Collection" for more information.

Topological Link Collection API Variation

Topological links are collected using different APIs depending on the vendor. Use one or both methods as required by the vendor or vendor device. Use the topologicalLinkCollectionType parameter to specify the collection method. See "APIs for Topological Link Collection" for more information.

Cross-Connect Protection Role

The Incremental TMF814 Discovery cartridge does not discover protection role information on cross-connect segments because vendors and devices differ in the way this information is accessed. You must extend the Incremental TMF814 Discovery cartridge to collect and model protection role information. See *Network Integrity Optical TMF814 CORBA Cartridge Guide* for more information.

5

About Cartridge Modeling

This chapter explains how the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge models collected data.

About Cartridge Modeling

The Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge models collected data according to the Oracle Communications Information Model. Collected data is modeled into the following entities:

- DeviceInterfaceConfiguration
- DeviceInterfaceConfigurationItem
- Equipment
- EquipmentHolder
- EquipmentEquipmentRel
- EquipmentHolderEquipmentRel
- InventoryGroup
- LogicalDevice
- MediaInterface
- PhysicalDevice
- PhysicalDeviceEquipmentRel
- PhysicalPort
- Pipe
- PipeTerminationPoint
- PipePipeTerminationPointRel

See *Oracle Communications Information Model Reference* for more information about the Information Model.

About the Oracle Communications Information Model

The Information Model has Physical and Logical Tree models. Physical device hierarchy is modeled in the Physical Tree. Logical device hierarchy is modeled in the Logical Tree.

This section details how the Multi Technology Network Management (MTNM) model is mapped to the Information Model.

About the Physical Tree

Table 5–1 shows how MTNM objects are mapped to Physical Tree entities.

Table 5–1 MTNM-to-Information Model Mapping for Physical Tree

MTNM Object	Information Model Entity	Specification
Manage Element (ME)	Physical Device	tmf814MEGeneric
Equipment Holder (Rack)	Equipment	tmf814EquipmentGeneric
Equipment Holder (Shelf)	Equipment	tmf814EquipmentGeneric A shelf is modeled as Equipment because the Information Model does not allow a holder within a holder.
Equipment Holder (sub Shelf)	Equipment	tmf814EquipmentGeneric
Equipment Holder (Slot)	Equipment Holder	tmf814EquipmentHolderGeneric
Equipment Holder (Sub Slot)	Equipment Holder	tmf814EquipmentHolderGeneric
Equipment (Card)	Equipment	tmf814EquipmentGeneric
Physical Termination Point (PTP)	Physical Port	tmf814PortGeneric
Topological Link	Pipe	tmf814TopologicalLinkGeneric
aEndTP, zEndTP (of a topological link object)	PipeTerminationPoint	tmf814PortTerminationPointGeneric
SNCCollection	Inventory Group	tmf814SNCGeneric
Cross-connect	InventoryGroup	tmf814XCGeneric
aEndName, zEndName (of a cross-connect	Pipe	tmf814XCSegmentGeneric A pair of related aEndName and zEndName objects are treated as a cross-connect segment.
aEndName, zEndName (of a cross-connect segment)	PipeTerminationPoint	tmf814PortTerminationPointGeneric

About the Logical Tree

Logical devices are created as root objects. Root objects are placeholder objects for top-level interfaces. PTPs and floating termination points (FTPs) are modeled as Device Interfaces. Contained termination points (TPs) of a PTP or FTP are modeled as sub-device-interfaces of a PTP or FTP device interface.

TPs that are discovered by the TMF814 API are modeled in the Logical Tree according to the following structure:

`Logical Device` (container for top level device interfaces){1}

  `Device Interface` (Device Interface corresponding to PTP/FTP) {0...*}

    `Sub Device Interface` (CTPs of PTP/FTP) {0...*}

      `Sub Device Interface` (child CTPs with any number of nesting levels) {0...*}

Layer parameters of a TP are modeled using the DeviceInterfaceConfigurationItem interface and its child interface configuration items. This cartridge models only Generally Applicable Parameters, which are defined and explained in the TMF814 documentation.

Each TP layer is represented by the DeviceInterfaceConfigurationItem interface. All TP layers are contained in an artificial parent DeviceInterfaceConfigurationItem interface, as shown in the following example:

`Device Interface` (represents a CTP/PTP/FTP)

`DeviceInterfaceConfigurationItem` (just a container configuration item){1}

`DeviceInterfaceConfigurationItem` (one configuration item per layer rate){0...*}

Table 5–2 shows how MTNM objects are mapped to Information Model entities in the Logical Tree.

Table 5–2 MTNM-to-Information Model Mapping for Logical Tree

MTNM Object	Information Model Entity	Specification
ME	LogicalDevice (artificial)	tmf814DeviceGeneric A logical device acts as a container for the top level interfaces. Its name is the same as the ME name.
PTP	DeviceInterface	tmf814TPInterfaceGeneric PTP as Interface is a container for the child CTP.
FTP	DeviceInterface	tmf814TPInterfaceGeneric FTP as Interface is a container for the child CTP.
Connection Termination Point (CTP)	DeviceInterface	tmf814TPLayersGeneric CTP is a channel and is modeled as a sub Device Interface.
LayeredParameters	DeviceInterfaceConfigurationItem	Managed Element Layered Parameters are modeled as configuration items of a Device Interface.

Field Mapping

The following tables explain the field mappings for each Information Model object.

- Table 5–3, " Physical Device Field Mapping"
- Table 5–4, " Equipment Field Mapping"
- Table 5–5, " Equipment Holder Field Mapping"
- Table 5–6, " Physical Port Field Mapping"
- Table 5–7, " Logical Device Field Mapping"
- Table 5–8, " Device Interface Field Mapping"
- Table 5–9, " Device Interface Configuration Item Field Mapping"

Table 5–3 Physical Device Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
Id	Static	N/A	Text	ID
name	Static	name	Text	Name
description	Static	N/A	Text	Description
specification	Static	N/A	PhysicalDeviceSpecification Programmatically set to the tmf814MEGeneric specification.	TMF814 MEGeneric
discoveredVendorName	Dynamic	manufacturer	Text Comes from additional information (not a TMF attribute).	Discovered Vendor Name
serialNumber	Static	N/A	Text	Serial Number
physicalLocation	Static	location	Text	Physical Location
softwareRev	Dynamic	version	Text	Software Version
modelName	Dynamic	productName	Text	Model Name
nativeEmsName	Static	nativeEmsName	Text	Native EMS Name
userLabel	Dynamic	userLabel	Text	Label
owner	Dynamic	owner	Text	Owner

Table 5–4 Equipment Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
Id	Static	N/A	N/A	ID
name	Static	name	Text	Name
description	Static	N/A	Text	Description
specification	Static	N/A	EquipmentSpecification Programmatically set to the tmf814EquipmentGeneric specification.	TMF814 Equipment Generic (displayed as Entity Type)
discoveredVendorName	Dynamic	manufacturer	Text Comes from additional information (not a TMF attribute).	Discovered Vendor Name
serialNumber	Static	installedSerialNumber	Text	Serial Number
physicalLocation	Static	N/A	Text	Physical Location
discoveredPartNumber	Dynamic	installedPartNumber	Text	Discovered Part Number
hardwareRev	Dynamic	installedVersion	Text	Hardware Rev
modelName	Dynamic	installedEquipmentObjectType	Text	Model Name

Table 5–4 (Cont.) Equipment Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
nativeEmsName	Static	nativeEmsName	Text	Native EMS Name
expectedObjectType	Dynamic	expectedEquipmentObjectType	Text	Expected Object Type
serviceState	Dynamic	serviceState	List: IN_SERVICE, OUT_OF_SERVICE, IN_MAINTENANCE, UNKNOWN, TESTING Each value corresponds to a TMF814 value: IN_SERVICE, OUT_OF_SERVICE, OUT_OF_SERVICE_BY_MAINTENANCE, and SERV_NA respectively. TMF814 does not have an equivalent for TESTING.	Service State
userLabel	Dynamic	userLabel	Text	Label
owner	Dynamic	owner	Text	Owner

Table 5–5 Equipment Holder Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
Id	Static	N/A	Text	ID
name	Static	name	Text	Name
description	Static	N/A	Text	Description
specification	Static	N/A	EquipmentHolderSpecification Programmatically set to the tmf814EquipmentHolderGeneric specification.	TMF814 Equipment Holder Generic (displayed as Entity Type)
serialNumber	Static	N/A	Text	Serial Number
physicalLocation	Static	N/A	Text	Physical Location
modelName	Dynamic	expectedOrInstalledEquipment	Text	Model Name
nativeEmsName	Static	nativeEmsName	Text	Native EMS Name
userLabel	Dynamic	userLabel	Text	Label
owner	Dynamic	owner	Text	Owner

Table 5–6 Physical Port Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
Id	Static	N/A	Text	ID
name	Static	name	Text /rack=1/shelf=1/slot=3/domain=sdh/port=1	Name
description	Static	N/A	Text	Description
specification	Static	N/A	PhysicalPortSpecification Programmatically set to tmf814PortGeneric specification.	TMF814 Port Generic (displayed as an Entity Type)
portNumber	Static	N/A	Integer	Port Number
customerPortName	Static	N/A	Text	Customer Port Name
vendorPortName	Static	N/A	Text	Vendor Port Name
serialNumber	Static	N/A	Text	Serial Number
physicalLocation	Static	N/A	Text	Physical Location
nativeEmsName	Static	N/A	Text	Native EMS Name
direction	Dynamic	direction	List: NA, BIDIRECTIONAL, SOURCE, SINK	Direction
tpProtectionAssociation	Dynamic	tpProtection Association	List: TPPA_NA, TPPA_PSR_RELATED	Protection Association
edgePoint	Dynamic	edgePoint	boolean	Edge Point
physicalAddress	Static	String	Text	Physical Address

Table 5–7 Logical Device Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
Id	Static	N/A	Text	ID
name	Static	name	Text	Name
description	Static	N/A	Text	Description
specification	Static	N/A	LogicalDeviceSpecification	TMF814 Device Generic (displayed as Entity Type)
nativeEmsAdmin ServiceState	Static	N/A	List: UNKNOWN, IN_SERVICE, OUT_OF_SERVICE, TESTING, IN_MAINTENANCE	Native EMS Admin Service State

Table 5–7 (Cont.) Logical Device Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
nativeEmsService State	Static	N/A	List: UNKNOWN, IN_SERVICE, OUT_OF_SERVICE, TESTING, IN_MAINTENANCE	Native EMS Service State
nativeEmsName	Static	nativeEms Name	Text	Native EMS Name
physicalLocation	Static	N/A	Text	Physical Location

Table 5–8 Device Interface Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
Id	Static	N/A	Text	ID
name	Static	name	Text	Name
description	Static	N/A	Text	Description
specification	Static	N/A	DeviceInterfaceSpecification Programmatically set to the tmf814TPInterfaceGeneric specification.	TMF 814 TPInterface Generic (displayed as Entity Type)
ifType	Static	Tp_type	List: CTP, PTP, FTP	Interface Type
interfaceNumber	Static	N/A	Text	Interface Number
customerInterfaceNumber	Static	N/A	Text	Customer Interface Number
vendorInterfaceNumber	Static	N/A	Text	Vendor Interface Number
nativeEmsName	Static	N/A	Text	Native EMS Name
nativeEmsAdminServiceState	Static	N/A	List: UNKNOWN, IN_SERVICE, OUT_OF_SERVICE, TESTING, IN_MAINTENANCE	Native EMS Admin Service State
nativeEmsServiceState	Static	N/A	List: UNKNOWN, IN_SERVICE, OUT_OF_SERVICE, TESTING, IN_MAINTENANCE	Native EMS Service State
mtuSupported	Static	N/A	Float	Supported MTU
mtuCurrent	Static	N/A	integer	Current MTU
physicalAddress	Static	N/A	Text	Physical Address
physicalLocation	Static	N/A	Text	Physical Location
minSpeed	Static	N/A	Float	Minimum Speed

Table 5–8 (Cont.) Device Interface Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
maxSpeed	Static	N/A	Float	Maximum Speed
nominalSpeed	Static	N/A	Float	Nominal Speed
connectionState	Dynamic	connectionState	List: TPCS_BI_CONNECTED, TPCS_NA, TPCS_SOURCE_CONNECTED, TPCS_SINK_CONNECTED, TPCS_BI_CONNECTED, TPCS_NOT_CONNECTED	Connection State
tpMappingMode	Dynamic	tpMappingMode	List: TM_NA (0), TM_NEITHER_TERMINATED_NOR_AVAILABLE_FOR_MAPPING (1), TM_TERMINATED_AND_AVAILABLE_FOR_MAPPING (2)	Termination Mode
direction	Dynamic	direction	List: NA, BIDIRECTIONAL, SOURCE, SINK	Direction
tpProtectionAssociation	Dynamic	tpProtectionAssociation	List: TPPA_NA, TPPA_PSR_RELATED	Protection Association
edgePoint	Dynamic	edgePoint	Boolean	Edge Point
userLabel	Dynamic	userLabel	Text	Label
owner	Dynamic	owner	Text	Owner
nativeEmsConnectorPresent	Static	N/A	Text	Native EMS Connector Present

Table 5–9 Device Interface Configuration Item Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
name	Static	N/A	Text Name is always set to LayerName	Name
value	Static	Layer	Text	Value
specification	Static	InventoryConfigurationSpec	Text Programmatically set to the tmf814TPLayersGeneric specification.	TMF814 TPLayer Generic (displayed as Entity Type)
clientType	Dynamic	clientType	Text	Client Type
potentialFutureSetupIndicator	Dynamic	potentialFutureSetupIndicator	List: RSU_POINT_TO_POINT, RSU_BROADCAST, RSU_ANY_CONFIG	Potential Future Setup Indicator

Table 5–9 (Cont.) Device Interface Configuration Item Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
serviceState	Dynamic	serviceState	List: IN_SERVICE, OUT_OF_SERVICE, IN_MAINTENANCE, UNKNOWN, TESTING Each value is mapped to TMF814 specific values: IN_SERVICE, OUT_OF_SERVICE, OUT_OF_SERVICE_BY_MAINTENANCE, and SERV_NA respectively. TMF814 does not have an equivalent for TESTING.	Service State
TCAParameterProfilePointer	Dynamic	TCAParameterProfilePointer	Text	TRA Parameter Profile Pointer
trailTraceExpectedRx	Dynamic	trailTraceExpectedRx	Text	Trail Trace Expected Rx
trailTraceMonitor	Dynamic	trailTraceMonitor	Text	Trail Trace Monitor
transmissionDescriptorPointer	Dynamic	transmissionDescriptorPointer	Text	Transmission Descriptor Pointer
allocatedNumber	Dynamic	allocatedNumber	Number	Allocated Number
dynamicAllocationEnabled	Dynamic	dynamicAllocationEnabled	Text	Dynamic Allocation Enabled

About Building the Information Model Tree

Collected TMF814 objects contain raw hierarchical details, but not at the object level. After the TMF814 objects are modeled as Information Model entities, they are added to the Physical or Logical Tree. This section describes the algorithm used for building the Trees.

Containment Relationships

To find containment relationship among discovered objects, the algorithm uses the Name attribute of TMF814 objects. The structure of the name is hierarchical and reflects the containment relationship between objects in a simple way. Table 5–10 describes the convention used for the field name.

Table 5–10 Name and Attribute Format for Containment Relationships

TMF Object	Name/Value Pairs
ME	name="EMS"; value="*CompanyName/EMSname*" name="ManagedElement"; value="*MEName*"
PTP	name="EMS"; value="*CompanyName/EMSname*" name="ManagedElement"; value="*MEName*" name="PTP"; value="*PTPName*"
FTP	name="EMS"; value="*CompanyName/EMSname*" name="ManagedElement"; value="*MEName*" name="FTP"; value="*FTPName*"

Table 5–10 (Cont.) Name and Attribute Format for Containment Relationships

TMF Object	Name/Value Pairs
CTP, as child of a PTP or FTP	name="EMS"; value="*CompanyName/EMSname*" name="ManagedElement"; value="*MEName*" name="PTP"; value="*PTPName*" name="CTP"; value="*CTPName*" name="FTP"; value="*FTPName*"
EquipmentHolder	name="EMS"; value="*CompanyName/EMSname*" name="ManagedElement"; value="*MEName*" name="EquipmentHolder"; value="*EquipmentHolderName*"
Equipment	name="EMS"; value="*CompanyName/EMSname*" name="ManagedElement"; value="*MEName*" name="EquipmentHolder"; value="*EquipmentHolderName*" name="Equipment"; value="*EquipmentName*"

The Equipment Holder tuple values are hierarchical and have the following structure:

```
[/remote_unit=<ru>][/rack=<r>][/shelf=<sh>[/sub_shelf=<ssh>][/slot=<sl>[/[remote_
]sub_slot=<ssl>]]]]
```

Adding an Equipment and an Equipment Holder to the Tree

The TMF814 Equipment Modeler processor is run for each EquipmentOrHolder TMF814 object. After modeling, the Equipment or Equipment Holder object is added to the Information Model Physical Tree.

It is possible that a child node can appear before its parent node is available. The algorithm handles this by using a placeholder node, which takes the place of the real node until the real node is available.

If the input object is a TMF814 Equipment Holder:

1. The EquipmentHolder tuple value is obtained from the name property. The tuple value is the hierarchical name of the Equipment Holder.
2. The name is split into two substrings at the last index of the / delimiter. This gives two placeholders:
   - The first placeholder gives the hierarchical name of the parent node, which is most likely another Equipment Holder.
   - The second placeholder is the shorter name for the Equipment Holder.

   ```
   index = lastIndexOf(name , "/");
   first = substring(name, 0, index)//First token
   second = substring(name, index +1, name.length)
   ```

3. If the first placeholder is empty, the Equipment Holder is a top-level object, and thus a parent node. The parent node is the node representing the physical device in the Tree.
4. If first placeholder is not empty, the Physical Tree is hierarchically searched from the root until the node representing the full hierarchical name is found. A placeholder is created for it while the Physical Tree is being searched.

   For example, if a placeholder is created for **/rack=1/shelf=2/slot=3**, it is split into **/rack=1**, **/rack=1/shelf=2**, and **/rack=1/shelf=2/slot=3**. The Physical Tree is

searched for **/rack=1**. If it is found, the search continues for **/rack=1/shelf=2**. If it is not found, a placeholder is created for it; **/rack=1/shelf=2/slot=3** is also not available, so a placeholder is created for it as well. The parent node is **/rack=1/shelf=2/slot=3**.

5. Parent nodes are verified to determine if they have any child nodes with a placeholder. If they do, the placeholder is released and is used for another node.
6. Nodes are created or replaced in the Physical Tree.

If the output object is TMF814 Equipment:

1. The EquipmentHolder tuple value is obtained from the name property.
2. The Physical Tree is hierarchically searched until the node representing the full hierarchical name is found. If the name is not found, a placeholder node is created for it.

   For example, if a placeholder is created for **/rack=1/shelf=2/slot=3**, it is split into **/rack=1**, **/rack=1/shelf=2**, and **/rack=1/shelf=2/slot=3**. The Physical Tree is searched for **/rack=1**. If it is found, the search continues for **/rack=1/shelf=2**. If it is not found, a placeholder is created for it; **/rack=1/shelf=2/slot=3** is also not available, so a placeholder is created for it as well. Parent node is **/rack=1/shelf=2/slot=3**.
3. Parent nodes are verified to determine if they have any child nodes with a placeholder. If they do, the placeholder is released and is used for another node.
4. Nodes are created or replaced in the Physical Tree.

After all nodes are modeled in the Physical Tree. Any remaining placeholder nodes are modeled as artificial objects.

Adding a Physical Port and an Interface to the Tree

TPs are modeled as physical ports. An associated artificial device interface is created for each physical port. A device interface is added as a direct child of a logical device.

The algorithm for adding equipment holders to the Tree can be applied to adding a physical port to the Physical Tree. See "Adding an Equipment and an Equipment Holder to the Tree" for more information.

Adding a Sub-Interface to the Tree

CTPs are modeled as Sub-Interfaces. They are added to the Logical Tree by the TMF814 CTP Discoverer for PTP and TMF814 CTP Discoverer for FTP processors, under the context of a PTP (top-level interface).

Cartridge Modeling for Cross-Connect Data

This section explains how the Incremental TMF814 Discovery cartridge models the collected cross-connect data.

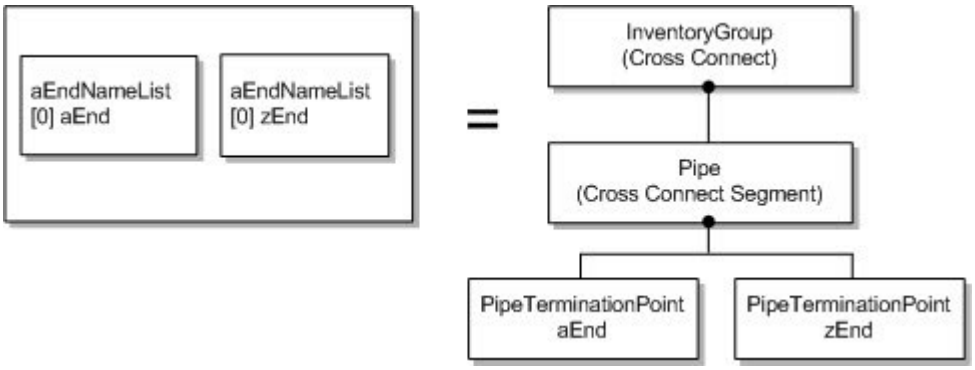
Only the cross-connect data required for assimilation is modeled. Of the data required for assimilation, only the data meeting the following conditions is modeled:

- At least one of the aEnd or zEnd TPs has a non-null or empty value.
- Both aEnd and zEnd represent CTP or FTP names.
- At least one of the aEnd or zEnd TPs has JKLM (VC12), JK (VC3), or J (VC4) values.

The Incremental TMF814 Discovery cartridge models cross-connects as one of the following types:

- ST_SIMPLE: Cross-connects with only one segment, as shown in Figure 5–1.

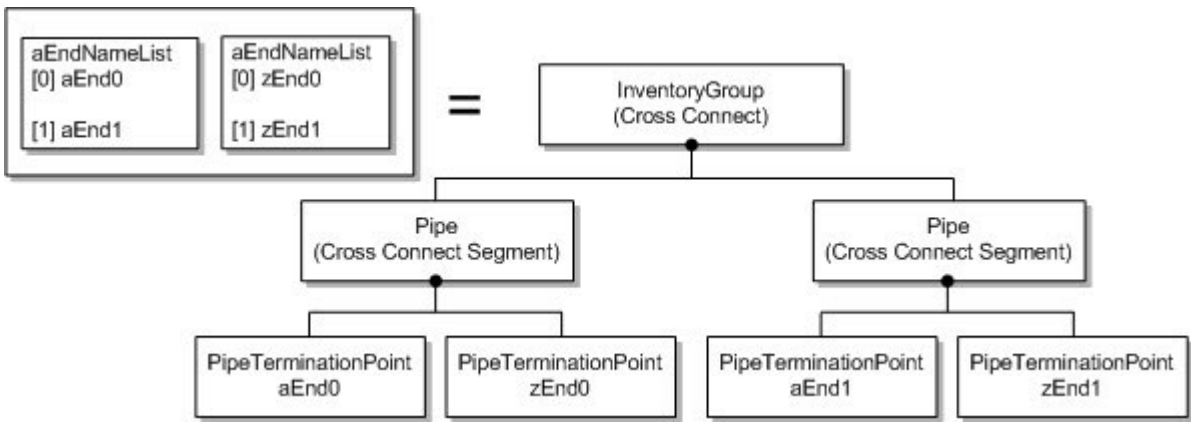
Figure 5–1 ST_SIMPLE Type Cross-Connect Model Mapping



Some vendors represent a bidirectional cross-connect as two unidirectional cross-connects, meaning one has A1-Z1 as its ends and other has Z1-A1 as its ends. Such cross-connects are modeled as bidirectional.

- ST_EXPLICIT: The cross-connect object is modeled as multiple pipe objects, as shown in Figure 5–2.

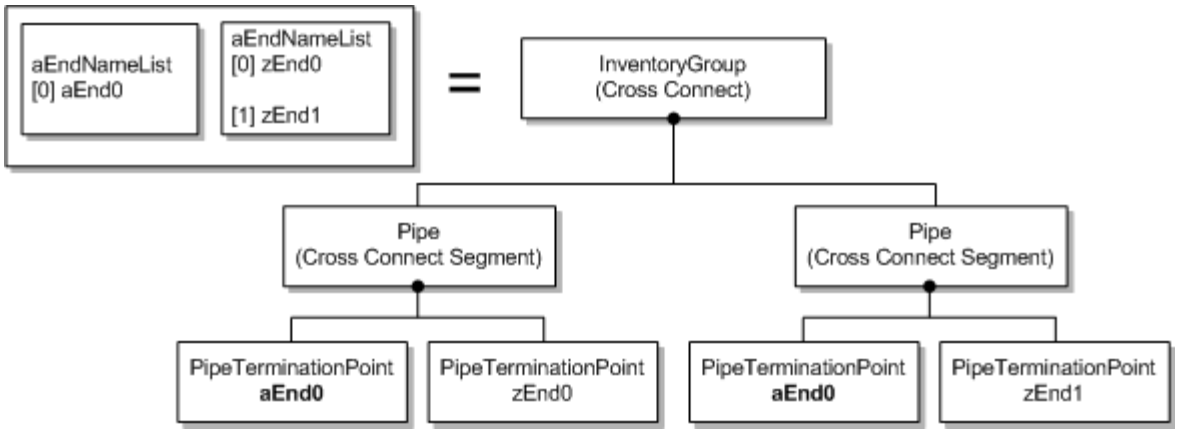
Figure 5–2 ST_EXPLICIT Type Cross-Connect Model Mapping



The number of objects into which a single cross-connect is modeled depends on aEndNameList and zEndNameList size. The explicit subnetwork connection (SNC) type has an *n*-entry aEndNameList and zEndNameList pairing. The tuples are pairs matched by index, for example (A1,Z1), (A2, Z2),...,(An,Zn). A pipe object is modeled for each pair. These multiple pipes are grouped by a parent Inventory Group object.

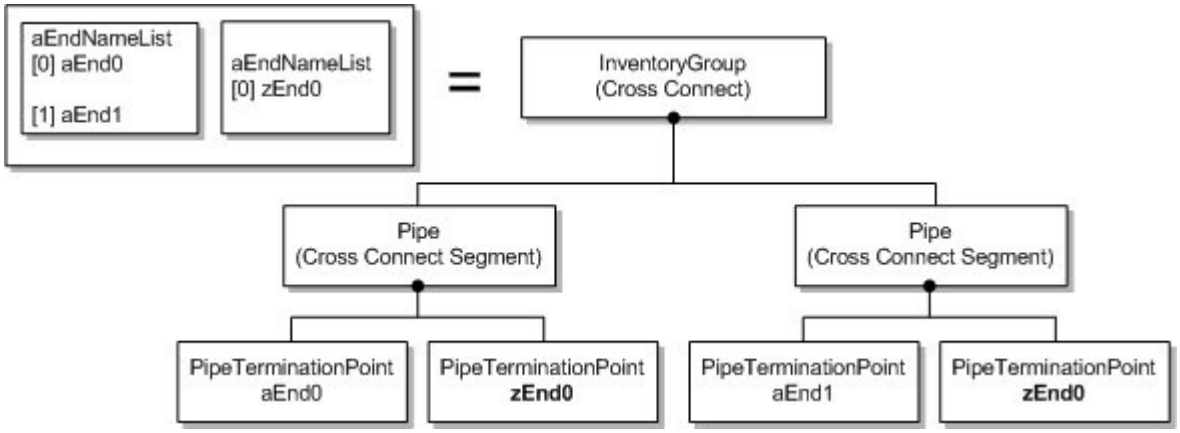
- ST_ADD_DROP_A: The cross-connect object is modeled as two pipe segments with aEndPoint repeating on both cross-connects segment, as shown in Figure 5–3.

Figure 5–3 ST_ADD_DROP_A Type Cross-Connect Model Mapping



- ST_ADD_DROP_Z: The cross-connect object is modeled as two pipe segments with zEndPoint repeating on both cross-connects segment, as shown in Figure 5–4.

Figure 5–4 ST_ADD_DROP_Z Type Cross-Connect Model Mapping



Other cross-connects types, such as ST_INTERCONNECT, ST_DOUBLE_INTERCONNECT, ST_DOUBLE_ADD_DROP, and ST_OPEN_ADD_DROP are not modeled by this cartridge without extending the cartridge.

The following tables list the model mapping of cross-connect objects:

- Table 5–11, " Model Mapping for the Inventory Group Object"
- Table 5–12, " Model Mapping for the Pipe Object"
- Table 5–13, " Model Mapping for the PipeTerminationPoint Object"

Table 5–11 Model Mapping for the Inventory Group Object

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
name	Static	N/A	Text The value is hard-coded as **Cross Connect**.	Name
layerRate	Dynamic	N/A	Text	Layer Rate
type	Dynamic	ccType	Text	Type
active	Dynamic	active	Text	Active

Table 5–12 Model Mapping for the Pipe Object

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
name	Static	N/A	Text	Name
gapPipe	Static	N/A	Boolean, always set to True.	Gap Pipe
protectionRole	Dynamic	N/A	Text The value is derived. Possible values are PRIMARY, BACKUP.	Protection Role

Table 5–13 Model Mapping for the PipeTerminationPoint Object

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
name	Static	N/A	Text The name of the PTP (port) cross-connect end point.	Name
device	Dynamic	N/A	Text	Device
directionality	Dynamic	N/A	Text	Directionality
rate	Dynamic	N/A	Text	Layer Rate
channel	Dynamic	N/A	Text Channel values are derived. See "A and Z Channels" for more information.	Channel

A and Z Channels

The following example SDH implementation shows how the channel is calculated for each PipeTerminationPoint.

Example CTP Name JKLM tuples:

- /sts3c_au4=4/vt2_tu12-k=1-l=3-m=2
- /direction=src/sts3c_au4=4/vt2_tu12-k=1-l=3-m=2
- /sts1_au3-j=2-k=2/vt15_tu11-l=1-m=2

JKLM values are collected from the *CTPName* tuple. Each CTP tuple can be split into a number of tokens separated by a slash. Each token can be further split into a number of subtokens separated by a hyphen.

If the *CTPName* tuple does not have any JKL or M value, it is treated as a dropdown port.

Example 5–1 shows how the JKLM values are parsed. This example assumes that the aEnd and zEnd of a cross-connect are a CTP with the formatting shown below:

Example 5–1 Parsed JKLM Values

```
Pattern pattern = Pattern.compile("/");
Matcher subTokenMatcher =
Pattern.compile("\\-j=\\d+|\\-k=\\d+|\\-l=\\d+|\\-m=\\d+").matcher("");
String STS3C_AU4 = "sts3c_au4=";

String[] jklm = new String[]{"0", "0", "0", "0"};
Scanner scaner = new Scanner(ctpName);
scaner.useDelimiter(pattern);
```

```
while(scaner.hasNext()){
    String token = scaner.next();
    subTokenMatcher.reset(token);
    while(subTokenMatcher.find()){
        String subToken = subTokenMatcher.group();
if(subToken.startsWith("-")){
            String val = token.substring(subTokenMatcher.start() +1,
subTokenMatcher.end());
            jklm[val.charAt(0) % 106] = val.substring(2, val.length());
        }else{
            jklm[subToken.charAt(0) % 106] = subToken.substring(2,
subToken.length());
        }
    }
    if(jklm[0].equalsIgnoreCase("0") && token.startsWith(STS3C_AU4)){
        jklm[0] = token.split("=")[1];;
    }
}
return jklm;
```

The Incremental TMF814 Discovery cartridge can be extended to populate JKLM values that are implemented differently by some vendors. See *Network Integrity Optical TMF814 CORBA Cartridge Guide* for more information.

Cartridge Modeling for Topological Link Data

This section explains how the Incremental TMF814 Discovery cartridge models collected topological link data.

Topological links are modeled Information Model pipe entities. Topological Link endpoints (aEndTP and zEndTP) are modeled as pipe termination point entities.

Some vendors represent bidirectional topological links as two unidirectional topological links (two links sharing the same aEnd and zEnd ports). Such links are merged and modeled as one bidirectional topological link.

The following tables list the model mapping of topological link objects:

- Table 5–14, " Model Mapping for the Pipe Object for Topological Links"
- Table 5–15, " Model Mapping for the PipeTerminationPoint Object for Topological Links"

Table 5–14 Model Mapping for the Pipe Object for Topological Links

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
name	Static	N/A	Text	Name
gapPipe	Static	N/A	Boolean This value is always set to False for topological link objects.	Gap Pipe
layerRate	Dynamic	rate	Text	Layer Rate
nativeEMSName	Dynamic	nativeEMSName	Text	Native EMS Name
owner	Dynamic	owner	Text	Owner

Table 5–15 Model Mapping for the PipeTerminationPoint Object for Topological Links

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
name	Static	name	Text	Name
device	Dynamic	N/A	Text The value is derived from the device.	Device
directionality	Dynamic	N/A	Text	Directionality
rate	Dynamic	N/A	Text This value is derived from the line layer rate for the endPort represented by the PortTerminationPoint.	Layer Rate
channel	Dynamic	N/A	Text This attribute is not used.	Channel

Result Groups

Topological link pipe entities are added to the device result group.

The cross-connect inventory group entities are added to the SNCCollection result group.

The SNC Collection result group is placed at the top-level along with the device result group and contains all SNCs. Each SNC contains the cross-connects belonging to that SNC.

Topological links span multiple devices. When the aEnd and zEnd ports are managed by MEs belonging to different EMSs, the topological link is modeled according to the device name that appears first in a sorted list.

The Link result group models a root entity container with the name Links as the parent for all topological links associated with a device. The topological link appears on the lower device of the two endpoints, as shown in Figure 5–5.

The cross-connect result group models a root entity container with the name Cross-connects as the parent for all cross-connects associated with the device, as shown in Figure 5–5.

Figure 5–5 Result Group Model Diagram

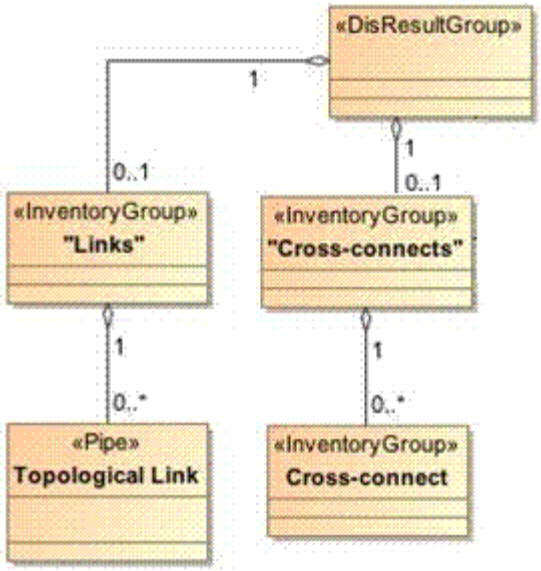
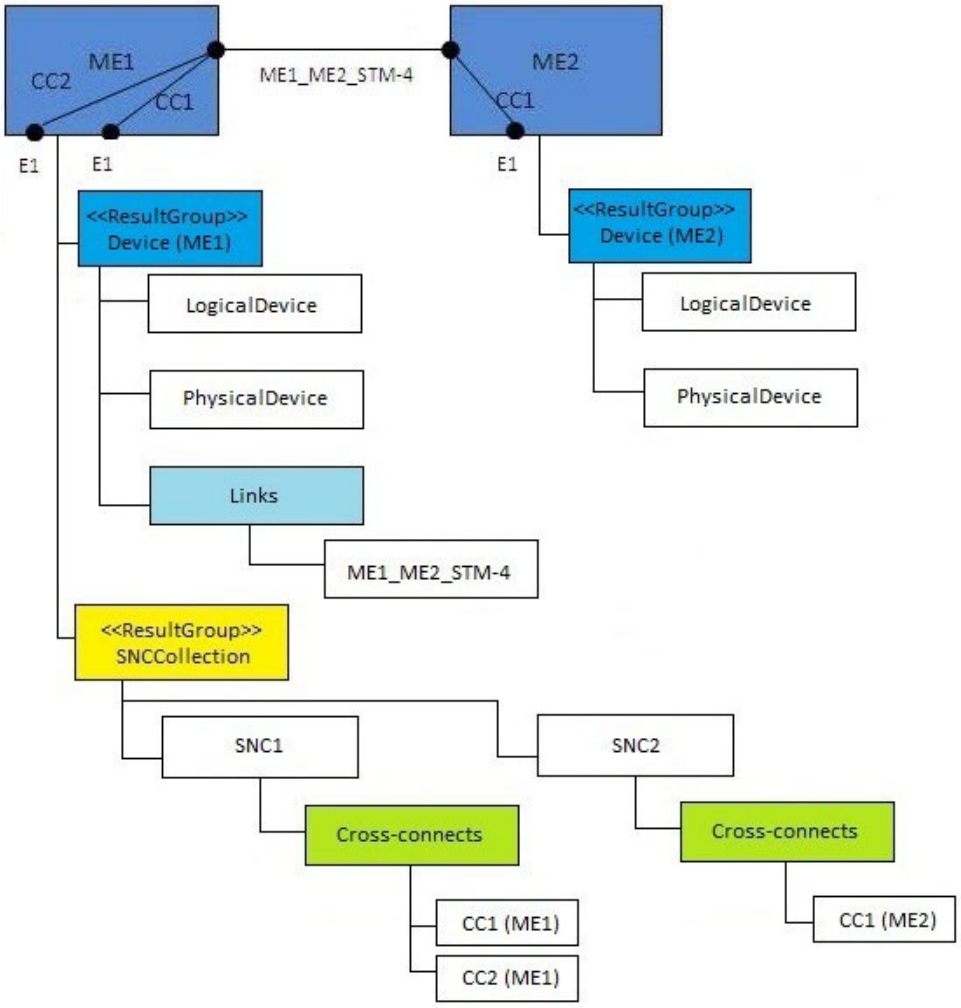


Figure 5–6 shows an example grouping for links and cross-connects with the following particularities:

- A populated result group for each device
- The appropriate cross-connects under different SNCs for each Subnetwork connection (SNC group)
- The topological link is added only to the ME1 device group

Figure 5–6 Example Result Group Model and Configuration



6

About Design Studio Construction

This chapter explains how the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge is built from the Oracle Communications Design Studio perspective.

Model Collections

Table 6–1 shows the Design Studio construction of the Generic TMF814 model collection.

Table 6–1 Generic TMF814 Model Collection

Specification Name	Dynamic Entity Type
tmf814MEGeneric	Physical Device Specification
tmf814DeviceGeneric	Logical Device Specification
tmf814EquipmentGeneric	Equipment Specification
tmf814EquipmentHolderGeneric	Equipment Holder Specification
tmf814PortGeneric	Physical Port Specification
tmf814TPInterfaceGeneric	Device Interface Specification This specification applies for all types of termination points (TPs).
tmf814TPLayersGeneric	Device Interface Configuration Specification

Actions

The following tables outline the Design Studio construction of the Incremental TMF814 Discovery cartridge actions and associated components:

- Table 6–2, " Actions Design Studio Construction"
- Table 6–3, " TMF814 Scan Parameters Design Studio Construction"
- Table 6–4, " Discovery Processor Design Studio Construction"

Note: Parameter values are case-sensitive and must be entered in capital letters when commands are run from a command line interface.

Table 6–2 Actions Design Studio Construction

Action Name	Result Category	Address Handler	Scan Parameter Group	Processors
Abstract Incremental Discovery action	Device	N/A	N/A	Incremental Discovery Initializer
Discover Incremental TMF814 action	Device	CorbaURLAddressHandler	TMF814Parameters. See Table 6–3. AutoResolutionParameter. See *Network Integrity Developer's Guide*. IncrementalScanParameter.	■ TMF814 Property Initializer ■ TMF814 Session Manager ■ TMF814 Device Recorder Initializer ■ Incremental Discovery Initializer ■ TMF814 ME Collector ■ TMF814 Updated ME Discoverer ■ TMF814 Device Modeler ■ TMF814 Equipment Collector ■ TMF814 Equipment Modeler ■ TMF814 PTP Collector ■ TMF814 PTP Modeler ■ TMF814 CTP Discoverer for PTP ■ TMF814 FTP Collector ■ TMF814 FTP Modeler ■ TMF814 CTP Discoverer for FTP ■ TMF814 Device Persister ■ TMF814 Device Recorder Persister ■ TMF814 Cross-Connect Discoverer ■ TMF814 SNC Discoverer ■ TMF814 SNC CC Discoverer ■ TMF814 Topological Link Collector ■ TMF814 Updated TL Discoverer ■ TMF814 Topological Link Modeler ■ TMF814 Pipe Persister

Table 6–3 TMF814 Scan Parameters Design Studio Construction

Parameter Name	Parameter Type	Description	UI Label
UserName	Text box	User name of the element management system (EMS) or network management system (NMS) used for getting details.	Username
Password	Secret text	Password of EMS or NMS system.	Password
EMSNamingService	Text box	EMS Naming Service The EMS session factory CORBA object name.	Ems Naming Service
EMSNamingServiceFormat	Drop down	List: PLAIN, STRINGIFIED The EMS session factory CORBA object name format.	Ems Naming Service Format

Table 6–3 (Cont.) TMF814 Scan Parameters Design Studio Construction

Parameter Name	Parameter Type	Description	UI Label
CollectEquipment	Drop down	List: TRUE, FALSE	Collect Equipment
CollectTP	Drop down	List: ALL, ONLY PTP, ONLY FTP, NONE	Collect Termination Points
CollectCTP	Drop down	List: CURRENT, IN USE, POTENTIAL, NONE	Collect Connection TP
EquipmentFetchSize	Text box	Number of equipment objects to fetch at a time for each EMS call.	Equipment Fetch Size
TPFetchSize	Text box	Number of contained TP objects to fetch at a time for each EMS call.	Termination Point Fetch Size
CTPCollectionDepth	Text box	The depth (level of children objects) to which contained TPs are collected.	Contained TP Collection Depth
ORBProperties	Text box	Semicolon separated name value pairs for ORB properties.	Orb Properties
ORBArguments	Text box	Semicolon separated name value pairs for ORB arguments.	Orb Arguments
ManagedElementName	Text box	Name of the ME. This parameter works in combination with Managed Element Name Qualifier. This parameter helps to filter the scan.	Managed Element Name
ManagedElementName Qualifier	Drop down	List: EQUALS, EQUALS_IGNORE_CASE, CONTAINS, CONTAINS_IGNORE_CASE, STARTS_WITH, STARTS_WITH_IGNORE_CASE, ENDS_WITH, ENDS_WITH_IGNORE_CASE This parameter works in combination with Managed Element Name to filter the collected MEs by name and qualifier.	Managed Element Name Qualifier
NetworkElementNames	Text box	Name of the NE. This parameter works in combination with Network Element Names Qualifier. This parameter helps to filter the scan.	Network Element Names
NetworkElementName Qualifier	Drop down	List: EQUALS, EQUALS_IGNORE_CASE, CONTAINS, CONTAINS_IGNORE_CASE, STARTS_WITH, STARTS_WITH_IGNORE_CASE, ENDS_WITH, ENDS_WITH_IGNORE_CASE COMMA_DELIMITED_NAMES, COMMA_DELIMITED_NAMES_IGNORE_CASE This parameter works in combination with Network Element Name to filter the collected NEs by name and qualifier.	Network Element Name Qualifier
CrossConnectCollection Type	Drop down	List: USE_SNC, USE_ME_MANAGER, NONE This parameter controls how cross-connects are collected. Select **None** to disable cross-connect collection.	Cross-connect Collection Type
TopologicalLinkCollectionType	Drop down	List: ALL, BETWEEN_SN, INSIDE_SN, NONE This parameter controls how topological links are collected. Select **None** to disable topological link collection.	Topological Link Collection Type

Table 6–4 Discovery Processor Design Studio Construction

Discovery Processor	Variable
TMF814 Property Initializer	Input: N/A Output: ■ tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties) A JavaBean that contains the set of TMF814 properties. See *Network Integrity Optical TMF814 CORBA Cartridge Guide* for a list of TMF814 Property Initializer properties and values. ■ tpDetailMap(oracle.communications.integrity.tmf814discovery.model.tp.TPNameMap) A map listing properties and associated PTP names.
TMF814 Session Manager	Input: tmf814Properties Output: ■ sessionManager(oracle.communications.integrity.tmf814discovery.session.SessionManager) A session manager instance responsible for creating emsMgr and multiLayerSubnetwork, and for managing EMSSession and TMF814 Object managers.
TMF814 Device Recorder Initializer	Input: tmf814Properties, customProperties Output: ■ recordMode(boolean) A Boolean indicating whether Recording Mode is enabled.
Incremental Discovery Initializer	Input: N/A Output: lastSuccessScanRun
TMF814 ME Collector	Input: N/A Output: ■ meIterable Iterable object for each collected ME.
TMF814 Updated ME Discoverer	Input: lastSuccessScanRun, meIterable Output: sncCollectionResultGrp
TMF814 Device Modeler	Input: ■ tmf814Properties ■ customProperties ■ managedElement(org.tmforum.mtnm.managedElement.ManagedElement_T) One instance of the meIterable object. This processor is run once for each instance of manamedElement. Output: ■ physicalTree(oracle.communications.integrity.tmf814discovery.model.ocimtree.PhysicalTree) A representation of the Information Model Physical Tree containing a physical device as the root object, to which child objects can be added. ■ logicalTree(oracle.communications.integrity.tmf814discovery.model.ocimtree.LogicalTree) A representation of the Information Model Logical Tree containing a logical device as the root object, to which child objects can be added.

Table 6–4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processor	Variable
TMF814 Equipment Collector	Input: tmf814Properties, customProperties, sessionManager, physicalTree, managedElement Output: ■ equipmentOrHolderIterable(java.lang.Iterable<org.tmforum.mtnm.equipment.EquipmentOrHolder_T>) Iterable object that iterates for each collected Equipment object or Holder object.
TMF814 Equipment Modeler	Input: ■ tmf814Properties ■ physicalTree ■ equipmentOrHolder(org.tmforum.mtnm.equipment.EquipmentOrHolder_T) One instance of the equipmentOrHolderIterable object. This processor is run once for each instance of equipmentOrHolder. Output: ■ equipment(oracle.communications.inventory.api.entity.Equipment) Returned value if the input is an equipment. ■ equipmentHolder(oracle.communications.inventory.api.entity.EquipmentHolder) Returned value if the input is an equipment holder.
TMF814 PTP Collector	Input: ■ tmf814Properties ■ customProperties ■ equipment(oracle.communications.inventory.api.entity.Equipment) A modeled Information Model equipment object. ■ equipmentHolder(oracle.communications.inventory.api.entity.EquipmentHolder) A modeled Information Model equipment holder object. Output: ■ ptpIterable(java.lang.Iterable<org.tmforum.mtnm.terminationPoint.TerminationPoint_T>) Iterable object for each collected PTP belonging to an Equipment object.
TMF814 PTP Modeler	Input: ■ tmf814Properties ■ equipment An Information Model object that is modeled as the parent for all ports. ■ physicalTree ■ logicalTree ■ tpDetailMap ■ ptp(org.tmforum.mtnm.terminationPoint.TerminationPoint_T) A PTP object, modeled as a Physical Port in the Physical Tree, and as a Device Interface in the Logical Tree. Output: ■ deviceInterface(oracle.communications.inventory.api.entity.DeviceInterface) A modeled Information Model interface object. ■ physicalPort(oracle.communications.inventory.api.entity.PhysicalPort) A modeled Information Model port object.

Table 6–4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processor	Variable
TMF814 CTP Discoverer for PTP	Input: ■ tmf814Properties Provides the CTP flag, termination point (TP) fetch size, and CTP depth properties. ■ customProperties ■ deviceInterface ■ logicalTree ■ physicalPort ■ ptp Parent PTP for which all CTPs are discovered. Output: N/A
TMF814 FTP Collector	Input: ■ tmf814Properties Provides the CTP flag and TP fetch size properties. ■ customProperties ■ logicalTree ■ managedElement The name of the ME is used to fetch the FTP. Output: ■ terminationPointIterable(java.lang.Iterable<org.tmforum.mtnm.terminationPoint.TerminationPoint_T>) Iterable object for each collected FTP.
TMF814 FTP Modeler	Input: ■ tmf814Properties ■ logicalTree ■ tpDetailMap ■ terminationPoint(org.tmforum.mtnm.terminationPoint.TerminationPoint_T) Output: ■ deviceInterface Modeled Information Model object for a TP. TPs are modeled and added to the Logical Tree as direct child objects of a logical device.
TMF814 CTP Discoverer for FTP	Input: ■ tmf814Properties ■ customProperties ■ terminationPoint TPs for which CTPs are fetched and modeled. ■ deviceInterface Parent Information Model object for all top level CTPs. ■ logicalTree Output: N/A

Table 6–4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processor	Variable
TMF814 Device Persister	Input: ■ tmf814Properties ■ physicalTree ■ logicalTree ■ managedElement Output: N/A
TMF814 Device Recorder Persister	Input: tmf814Properties Output: N/A
TMF814 Cross-Connect Discoverer	Input: tmf814Properties, customProperties, tpDetailMap Output: N/A
TMF814 SNC Discoverer	Input: customProperties, lastSuccessScanRun, sncCollectionResultGrp, tmf814Properties Output: emsName, sncList
TMF814 SNC CC Discoverer	Input: customProperties, lastSuccessScanRun, snc, tpDetailsMap, tmfNameToDeviceNameMap tmf814Properties Output: N/A
TMF Topological Link Collector	Input: tmf814Properties, customProperties, tpDetailMap Output: ■ topologicalLinkIterable(java.util.Iterable) Iterable object that iterates for each collected topological link object. ■ tlPipeMap(java.util.Map<java.lang.String,java.utilList<oracle.communications.inventory.api.entity.Pipe>>) A map listing all collected topological links by their container group.
TMF814 Updated TL Discoverer	Input: emsName, tpDetailsMap, lastSuccessScanRun, tlPipeMap, tmf814Properties, topologicalLinkIterable Output: N/A
TMF814 Topological Link Modeler	Input: ■ tmf814Properties ■ tpDetailMap ■ topologicalLink(org.tmforum.mtnm.topologicalLink.TopologicalLink_T) ■ tlPipeMap Output: ■ linkPipe(oracle.communications.inventory.api.entity.Pipe) A modeled topological link as a pipe entity.
TMF814 Pipe Persister	Input: tmf814Properties, tpDetailMap, tlPipeMap Output: N/A

7

About Design Studio Extension

This chapter contains examples and explanations of how to extend certain aspects of the Oracle Communications Network Integrity Incremental TMF814 Discovery cartridge by using Oracle Communications Design Studio. See *Network Integrity Developer's Guide* for more information. See *Network Integrity Concepts* for guidelines and best practices for extending cartridges.

The following examples are explained in this section:

- Initializing a Custom Object Request Broker
- Extending the Discover Incremental TMF814 Action

Initializing a Custom Object Request Broker

This example explains how you can initialize a custom object request broker (ORB) instead of using the default ORB provided by the Network Integrity Cartridge for CORBA (CORBA cartridge).

To initialize a custom ORB:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the CORBA cartridge project.
4. Create a discovery action that uses the CORBA Abstract Discovery action as a processor.
5. Create a discovery processor named Custom CORBA Property Initializer and add it after the CORBA Property Initializer processor.

   This processor overrides org.omg.CORBA.ORBClass, org.omg.CORBA.ORBSingletonClass, and any additional parameters specific to ORB implementation from the CORBAProperties JavaBean.
6. Create a discovery processor named Custom ORB Manager to perform custom lookup.
7. (Optional) Disable NamingContextExt lookup.

   When enabled, this operation may set the Naming Service Connection Flag to false, causing custom lookup to fail.

Extending the Discover Incremental TMF814 Action

This example explains how to extend the Discover Incremental TMF814 action to collect vendor-specific information. No new common object request broker architecture (CORBA) calls are required to the server because this data is already collected. In this example, managementIP of a managed element (ME) is used as the desired vendor-specific information.

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project named TMF814SampleVendorExtension.
3. Make TMF814SampleVendorExtension dependent on the Incremental TMF814 Discovery cartridge project and the TMF814_Model cartridge.
4. Create a discovery action named TMF814 Sample Vendor Extension.
5. Add the Discover TMF814 action as a processor in the TMF814 Sample Vendor Extension action.
6. Create a Physical Device specification named customTMF814MEGeneric and add all the same characteristics as tmf814MEGeneric.
7. Add the managementIP characteristic to customTMF814MEGeneric.
8. Create a new discovery processor named Custom Device Modeler and insert it after the TMF814 Device Modeler processor. Specify **physical device** and **managed element** as input parameters to the processor.
9. In the Custom Device Modeler processor implementation, add code to populate the physical device with managementIP, as shown in the following example:

```
//Get ME form request
ManagedElement_T me = request.getManagedElement();
PhysicalDevice dev = request.getPhysicalDevice();

//Create CustomTMF814MEGeneric spec isntance
CustomTMF814MEGeneric customDevice = new CustomTMF814MEGeneric(dev);

//TMFAdditionalInfoHelper is a helper class bundled with this cartridge

String managementIP = TMFAdditionalInfoHelper.getAdditionalInfo
(me.additionalInfo, "managementIP");

customDevice.setManagementIP(managementIP);
```

10. Build, deploy, and test your cartridge.

    Your new Custom Device Modeler processor is run in the order shown in Figure 7–1.

Figure 7–1 Custom Device Modeler Processor Workflow

8

NMS Notification Listener Sample Application Reference

This chapter provides information about the example code for the NMS Notification Listener Sample Application.

Configuring the NMS Notification Listener Sample Application to Receive Notifications

To enable the Notification Listener Sample Application to connect to the NMS Server and receive notification events, configure the Sample Application by doing the following:

1. Getting the ORB Object
2. Getting the EMS Session Object
3. Getting the Event Channel Object from the EMS Session Object
4. Getting the Consumer Admin Object from the Event Channel Object
5. Connecting to the StructuredProxyPullSupplier from the Consumer Admin Object
6. Pulling the Notification Events from StructuredProxyPullSupplier
7. Saving the Generated Notification Events in XML Files

The following subsections show example Java code for the notification listener, which connects to the EMS or NMS servers and receives notifications. You can extend the example Java code based on the EMS or NMS you are using. The EMS or NMS saves its notification events in the StructuredProxyPullSupplier object.

Getting the ORB Object

```
String nameService = "NameService="+corbaUrl;
String[] orbArgs =  new String[] {"-ORBInitRef", nameService};
ORB orb = ORB.init(orbArgs, null );
```

Getting the EMS Session Object

```
EmsSession_I emsSession =  null;
NamingContextExt ncRef = null;
try {
                  ncRef NamingContextExtHelper.narrow(orb.resolve_initial_
references("NameService"));
} catch (InvalidName e) {
```

```
    e.printStackTrace();
    throw e;
}
NameComponent tmfClass = new NameComponent(nmsDetails.getClazz(), "Class");
NameComponent tmfVendor = new NameComponent(nmsDetails.getVendor(), "Vendor");
NameComponent tmfEmsInstance = new
NameComponent(nmsDetails.getEmsInstance(),"EmsInstance");
NameComponent tmfVersion = new NameComponent(nmsDetails.getVersion(), "Version");
NameComponent tmfEntity = new
NameComponent(nmsDetails.getEmsSessionFactory(),"EmsSessionFactory_I");
NameComponent[] name = {tmfClass, tmfVendor, tmfEmsInstance, tmfVersion,
tmfEntity};
EmsSessionFactory_I sessionFactory = null;
try {
    sessionFactory = EmsSessionFactory_IHelper.narrow(ncRef.resolve(name));
} catch (NotFound e) {
    throw e;
} catch (CannotProceed e) {
    throw e;
} catch (org.omg.CosNaming.NamingContextPackage.InvalidName e) {
    throw e;
}
POA rootpoa = null;
try {
    rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
} catch (InvalidName e) {
    throw e;
}
try {
    rootpoa.the_POAManager().activate();
} catch (AdapterInactive e) {
    throw e;
}
NmsSessionImpl nmsSessionImpl = new NmsSessionImpl(orb);
org.omg.CORBA.Object corbaObj = null;
try {
    corbaObj = rootpoa.servant_to_reference(nmsSessionImpl);
} catch (ServantNotActive e) {
    throw e;
} catch (WrongPolicy e) {
    throw e;
}
NmsSession_I nmsSession = NmsSession_IHelper.narrow(corbaObj);
EmsSession_IHolder sessionHolder = new EmsSession_IHolder();
// Read the NMS User credentials from the console.
Console console = System.console();
String username = console.readLine(sub_nl +" ["+ nmsDetails.getEmsInstance() + "]
Login as: ");
char[] password = console.readPassword(sub_nl +" ["+ nmsDetails.getEmsInstance()
+"] password: ");
String passwd = new String(password);
try {
    sessionFactory.getEmsSession(username, passwd, nmsSession, sessionHolder);
} catch (ProcessingFailureException e) {
    throw e;
}
emsSession =  sessionHolder.value;
```

Getting the Event Channel Object from the EMS Session Object

```
EventChannelHolder eventChannelHolder = new EventChannelHolder();
try {
emsSession.getEventChannel(eventChannelHolder);
} catch (ProcessingFailureException e) {
    throw e;
}
EventChannel eventChannel = eventChannelHolder.value;
```

Getting the Consumer Admin Object from the Event Channel Object

```
// In  NMS Notification Service, if only one Customer Admin Object is configured,
// obtain the ConsumerAdmin Object by calling default_consumer_admin().
ConsumerAdmin consumerAdmin = eventChannel.default_consumer_admin();
// otherwise
// int consumerAdminId = 0; // Get the consumer Admin Id from NMS Notification
// configuration details.
    //try {
    //     return eventChannel.get_consumeradmin(consumerAdminId);
    //} catch (AdminNotFound e) {
    //     e.printStackTrace();
//}
```

Connecting to the StructuredProxyPullSupplier from the Consumer Admin Object

```
org.omg.CORBA.IntHolder proxyId = new org.omg.CORBA.IntHolder();
ProxySupplier proxySupplier = null;
try {
proxySupplier = consumerAdmin.obtain_notification_pull_
supplier(ClientType.STRUCTURED_EVENT, proxyId); // In cases where different
// ClientType is configured on NMS, change the ClientType to obtain
// appropriateProxyPullSupplier.
    }
catch ( org.omg.CosNotifyChannelAdmin.AdminLimitExceeded ex ) {
    ex.printStackTrace( System.err );
    System.exit(-1);
}
StructuredProxyPullSupplier s_proxy_pull_supplier = null;
try {
    s_proxy_pull_supplier =
StructuredProxyPullSupplierHelper.narrow(proxySupplier);
}
catch ( org.omg.CORBA.BAD_PARAM ex ) {
    ex.printStackTrace( System.err );
    System.exit(-1);
}
try {
    s_proxy_pull_supplier.connect_structured_pull_consumer( null );
}
```

Pulling the Notification Events from StructuredProxyPullSupplier

```
/**
     * This method pulls the Notification Events from NMS
StructuredProxyPullSupplier.
     */
```

```
    public void startPullEvents() {
        System.out.println("Pulling the events..." );
        EventPersister persister = new EventPersister(eventQueue);
        persister.start();
        if(proxyPullSupplier != null) {
            while ( true ) {
                try {
                    StructuredEvent event proxyPullSupplier.pull_structured_
event();
                    if(event != null){
                         eventQueue.add(event);
                    }
                }
                catch ( org.omg.CosEventComm.Disconnected ex ) {
                    ex.printStackTrace( System.err );
                    return;
                }
            }
        }
    }
```

Saving the Generated Notification Events in XML Files

The NMS Notification Listener Sample Application creates the following XML files to store details for each of the following types of entities:

1. Create the *EMS_Name*.**me** file (where *EMS_Name* is the name of the EMS) to contain the list of details about managed elements (MEs) on the EMS or NMS.
2. Create the *EMS_Name*.**tl** file to contain the list of details about topological links (TLs) on the EMS or NMS.
3. Create the *EMS_Name*.**snc** file to contain the list of details about subnetwork connections (SNCs) on the EMS or NMS.
4. Create the *EMS_Name*.**deted_tl** file to contain the list of deleted TLs on the EMS or NMS.

```
/**
* The persister saves the last modified information about the MEs, SNCs, and TLs
in XML files.
*/
class EventPersister extends Thread {
    private Queue<StructuredEvent> eventQueue;
    public EventPersister(Queue<StructuredEvent> eventQueue){
           this.eventQueue = eventQueue;
    }
    public void run(){
    while(true){
    if(!eventQueue.isEmpty()){
    while(!eventQueue.isEmpty()){
       // TODO: record the notifications into files based on ObjectType.
try {
                  StructuredEvent event = eventQueue.poll();
Property[] properties = event.filterable_data;
// Get the Object Name on which the notification is generated.
NameAndStringValue_T[] objectName = NVSList_THelper.extract(properties[1].value);
// Get the Object Type on which the notification is generated.
ObjectType_T objectType = ObjectType_THelper.extract(properties[2].value);
// Get the EMS Time at which the notification is generated.
```

```
// Formatted in yyyyMMddHHmmss
String emsTime = properties[3].value.toString();
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMddHHmmss");
String emsName = objectName[0].value;
String meName = null;
String sncName = null;
String topologicalLinkName = null;
System.out.println("EMS ["+emsName+"]");
if(objectType == ObjectType_T.OT_MANAGED_ELEMENT || objectType == ObjectType_T.OT_
EQUIPMENT ||
objectType == ObjectType_T.OT_EQUIPMENT_HOLDER || objectType == ObjectType_T.OT_
PHYSICAL_TERMINATION_POINT) {
meName = objectName[1].value;
System.out.println("ME ["+meName+"] LastModified:
["+dateFormat.parse(emsTime)+"]");
} else if(objectType == ObjectType_T.OT_TOPOLOGICAL_LINK){
    topologicalLinkName = objectName[1].value;
    System.out.println("TL ["+topologicalLinkName+"] LastModified:
["+dateFormat.parse(emsTime)+"]");
} else if(objectType == ObjectType_T.OT_SUBNETWORK_CONNECTION){
    sncName = objectName[2].value;
    System.out.println("SNC ["+sncName+"] LastModified:
["+dateFormat.parse(emsTime)+"]");
}
} catch(Exception ex){
ex.printStackTrace();
}
    }
    }
    }
}


/* Sample Notification Generated on Adding New EquipmentHolder.
           ************************************************
           START OF NT_OBJECT_CREATION
           ************************************************
           name: notificationId value:29
           name: objectName value:
                        name: EMS value:ECI/LightSoft_1
                        name: ManagedElement value:LSN/EMS_XDM_121/1064
                        name: EquipmentHolder value:/rack=1/shelf=1/slot=1/sub_
slot=4
           name: objectType value:OT_EQUIPMENT_HOLDER
           name: emsTime value:20140302141559.0
           name: neTime value:
           name: edgePointRelated value:FALSE
           name: remainder_of_body  value:
                   Name: attributeList value:
                        name: EMS value:ECI/LightSoft_1

                        name: ManagedElement value:LSN/EMS_XDM_121/1064
                        name: EquipmentHolder value:/rack=1/shelf=1/slot=1/sub_
slot=4

                   name: userLabel value:I1 OTR1 4
                   name: nativeEMSName value:I1 OTR1 4
                   name: owner value:
                   name: alarmReportingIndicator value:TRUE
                   name: holderType value:sub_slot
```

```
                        name: expectedOrInstalledEquipment value:
                                name: EMS value:LSN/EMS_XDM_121

                                name: ManagedElement value:1064
                                name: EquipmentHolder value:/rack=1/shelf=1/slot=1/sub_
slot=4

                                name: Equipment value:1
                        name: EquipmentObjectTypeList value:
                                NONE
                                ETR1
                                OTR1
                        name: holderState value:EXPECTED_AND_NOT_INSTALLED
                        name: additionalInfo value:
                                name: LSNExt_TimestampSignature value:20140302141559.8

                ************************************************************
                END OF NT_OBJECT_CREATION
                ************************************************************
                */
    }
```

Extending the NMS Notification Listener Sample Application to Filter Notifications

To receive notification messages, you must configure the EMS or NMS to generate different types of notification events, such as SYSLOG, ALARMS, and so on. The NMS Notification Listener Sample Application does not have any filters that restrict notifications from the EMS or NMS and hence it receives all types of notification messages. You can extend the Sample Application to filter notifications based on your business requirements. You add notification filters by extending the StructuredPushConsumer interface that is part of the Sample Application.

You can filter events on both the supplier and consumer side. In particular, filters may be applied to the supplier and consumer admins and to supplier and consumer proxies.

To apply a filter to the supplier admin or proxy, extend the StructuredPushConsumer interface by doing the following:

1. Obtaining a Reference to the Filter Factory
2. Creating a Filter
3. Adding the Filter to an Admin or Proxy

The following subsections show example Java code, which you can extend to filter different types of notifications events.

Obtaining a Reference to the Filter Factory

Applying a filter means obtaining a reference to the Default Filter Factory. Note that every object of type CosNotifyChannelAdmin:: EventChannel includes a reference to the DefaultFilterFactory.

```
FilterFactory filterFactory = eventChannel.default_filter_factory();
```

Creating a Filter

Do the following to create a simple filter, as implemented by the `FilteredConsumer` class:

Create Constraints

```
final int numberofConstraints = 4;
String[] constraintStrings =
{

"$domain_name == Telecom",
"$type_name == CommunicationsAlarm",
"$type_name == Notification",
"$severity != 4"

};

ConstraintExp[] constraints =  new ConstraintExp[numberofConstraints];

for(int i = 0 ; i < numberofConstraints ; ++i)
{
EventType eventType = new EventType();
eventType.domain_name = "*";
eventType.type_name = "*";

EventType[] eventTypes = new EventType[1];
eventTypes[0] = eventType;

ConstraintExp constraint = new ConstraintExp();
constraint.event_types = eventTypes;
constraint.constraint_expr = constraintStrings[i];

constraints[i] = constraint;
}
```

Create a Filter

Call the create_filter() method on the FilterFactory to create a filter.

```
filter = filterFactory.create_filter("EXTENDED_TCL");
```

where:

EXTENDED_TCL is the default grammar supported by all compliant notification services.

Add Constraints to the Filter

Add the constraints you created to the filter by calling the add_constraints() method.

```
ConstraintInfo[] info = filter.add_constraints(constraints);
```

Adding the Filter to an Admin or Proxy

The following interfaces are inherited from the FilterAdmin interface and can have filter objects added to them:

```
ProxyConsumer
ProxySupplier
ConsumerAdmin
```

```
SupplierAdmin
```

For example, the following line adds a filter to a supplier proxy:

```
proxySupplier.add_filter(filter);
```