Oracle® Communications Network Integrity

Optical TMF814 CORBA Cartridge Guide

Release 7.4
F93128-01
July 2024

ORACLE

Oracle Communications Network Integrity Optical TMF814 CORBA Cartridge Guide, Release 7.4

F93128-01

Contents

Preface

This guide explains the functionality and design of the Oracle Communications Network Integrity Optical TMF814 CORBA 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)

This guide assumes that you 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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Overview

This chapter describes the Oracle Communications Network Integrity Optical TMF814 CORBA cartridge.

About the Optical TMF814 CORBA Cartridge

The Optical TMF814 CORBA cartridge is used to discover your 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 managed elements (MEs). It uses the TMF814 CORBA interface as a discovery protocol to connect and retrieve details from network management systems (NMSs) or element management systems (EMSs).

Using this cartridge, you can configure Network Integrity to capture and retrieve data about a network system from equipment and system vendors that have adopted the TMF814 standard.

The Optical TMF814 CORBA 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

This cartridge supports versions 2.0, 2.1, 3.0, and 3.2 of the TMF814 implementation for the ManagedElementMgr and EquipmentInventoryMgr managers.

This cartridge translates MTNM objects obtained during discovery into the Oracle Communications Information Model and then writes the objects to the Network Integrity database.

To ensure scalability, this cartridge processes MEs individually. The duration of the discovery actions is proportional to the number and size of MEs to be discovered. It is not possible to pause and resume a scan, though a scan can be stopped.

About Cartridge Dependencies

This section provides information on dependencies that the Oracle Communications Network Integrity Optical TMF814 CORBA cartridge has on other entities.

Run-Time Dependencies

There are no run-time dependencies for this cartridge.

Design Studio Dependencies

To load the Optical TMF814 CORBA cartridge into Oracle Communications Design Studio, the following cartridge must be installed:

- Network Integrity cartridge for CORBA (CORBA cartridge), including all of its dependencies

Opening the Cartridge Files in Design Studio

To review and extend the Optical TMF814 CORBA cartridge, download a ZIP file from the Oracle software delivery web site:

https://edelivery.oracle.com/

Download the Optical TMF814 CORBA cartridge ZIP file for Network Integrity, which contains the Design Studio cartridge files.

The Optical TMF814 CORBA cartridge ZIP file has the following structure:

- **\UIM_Cartridge_Projects\ora_ni_uim_ocim**
- **\UIM_Cartridge_Projects\TMF814_Model**
- **\Network_Integrity_Cartridge_Projects\TMF814Discovery_Cartridge**
- **\Network_Integrity_Cartridge_Projects\Abstract_CORBA_Cartridge**

The **TMF814Discovery_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 Optical TMF814 CORBA cartridge.

The Optical TMF814 CORBA cartridge contains the following actions:

- Discover Abstract TMF814 Action
- Discover TMF814 Action

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

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

Discover Abstract TMF814 Action

This is an abstract action that can be extended in Oracle Communications Design Studio to discover specified network and connectivity objects, using specified ORB Properties and ORB Arguments. This action uses the **IncrementalScanParameters** scan parameter group. This action supports incremental discovery based on NMS notifications received by Network Integrity.

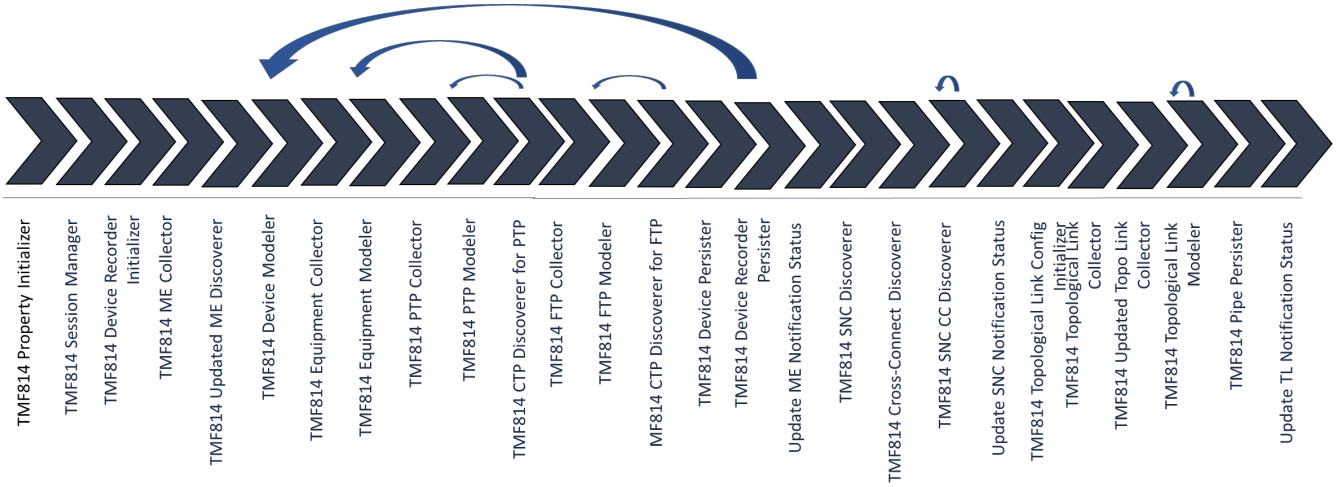
The Discover Abstract TMF814 action contains the following processors run in the following order:

1. TMF814 Property Initializer
2. TMF814 Session Manager
3. TMF814 Device Recorder Initializer
4. TMF814 ME Collector
5. TMF814 Updated ME Discoverer
6. TMF814 Device Modeler
7. TMF814 Equipment Collector
8. TMF814 Equipment Modeler
9. TMF814 PTP Collector
10. TMF814 PTP Modeler
11. TMF814 CTP Discoverer for PTP
12. TMF814 FTP Collector
13. TMF814 FTP Modeler
14. TMF814 CTP Discoverer for FTP
15. TMF814 Device Persister
16. TMF814 Device Recorder Persister
17. Update ME Notification Status

18. TMF814 SNC Discoverer
19. TMF814 Cross-Connect Discoverer
20. TMF814 SNC CC Discoverer
21. Update SNC Notification Status
22. TMF814 Topological Link Config Initializer
23. TMF814 Topological Link Collector
24. TMF814 Updated Topo Link Collector
25. TMF814 Topological Link Modeler
26. TMF814 Pipe Persister
27. Update TL Notification Status

Figure 2-1 illustrates the processors workflow of the Discover Abstract TMF814 action.

Figure 2-1 Discover Abstract TMF814 Action Processors



> **Note:**
>
> The Abstract TMF814 action is configured with the **isIncrementalScanEnabled** condition. When this condition is met, the corresponding processors will run.

TMF814 Property Initializer

This processor initializes properties required by other processors in the Discover Abstract TMF814 action. All properties are populated into a JavaBean class named tmf814Properties. These properties can be customized by other processors. Table 2-1 lists the available properties.

Table 2-1 TMF814 Property Initializer Properties and Values

Property Name	Description	Value
Username	Indicates the EMS user name, used while getting the EMS session from the session factory.	Extending cartridges may supply a default value.
Password	Indicates the EMS password, used while getting the EMS session from the session factory.	Extending cartridges may supply a default value.
collectCTP	Indicates the type of CTPs to collect: • A current TP is a CTP that is, or can be cross-connected in the current mapping configuration. • An in-use TP is a CTP that is used by a subnetwork connection (SNC) in any state, or a CTP that is terminated and mapped. • A potential contained TP (contained in a PTP or CTP) is a CTP that is capable of supporting all possible mapping configurations at the specified layer rates and is contained by the specified termination.	Valid values: NONE, CURRENT, IN_USE, POTENTIAL
collectEquipment	Indicates whether to collect Equipment and Equipment Holder details. If this value is FALSE, only MEs and connection termination points (CTPs) associated with floating termination points (FTPs) are collected and modeled.	Valid values: TRUE, FALSE
collectTP	Indicates whether to collect physical termination points (PTPs) or floating termination points (FTPs), or both.	Valid values: ALL, NONE, ONLY_PTP, ONLY_FTP
crossConnectCollectionType	Indicates the method used to collect cross-connects: • A value of **USE_SNC** collects cross-connects using multi layer subnetwork (MLSN) Manager APIs. • A value of **USE_ME_MANAGER** collects cross-connects using ME Manager APIs. Cross-connects are not collected if the value is set to **NONE**. Ensure the correct value is used, according to your vendor specifications.	Valid values: USE_SNC, USE_ME_MANAGER, NONE
crossConnectModelCustomizerImplClass	Allows you to customize cross-connect modeling.	N/A
ctpCollectionDepth	Indicates the hierarchical depth to which CTPs are collected (because TPs can contain several levels of child TPs), depending on the API used by the vendor to call CTPs.	Valid values: positive integers
ctpModelCustomizerImplClass	A class implementing the oracle.communications.integrity.tmf814discovery.model.ctp.CTPModelCustomizer interface. To discover additional attributes for CTPs, add a similar implementation interface to the system.	N/A
discovererFactoryImplClass	A default implementation of the Discoverer Factory class, used to provide a custom collection mechanism.	N/A
emsManagerName	Name of the EMSMgr_I manager used to obtain the manager from EmsSession_I.getManager.	N/A
equipmentFetchSize	Indicates the number of Equipment or Holders to fetch at a time.	Valid values: positive integer
equipmentInventoryManagerName	Indicates the name of the Equipment inventory manager.	N/A
includeHigherOrderCCs	Specifies whether higher-order cross-connects of other SNCs are collected.	Valid values: TRUE, FALSE

Table 2-1 (Cont.) TMF814 Property Initializer Properties and Values

Property Name	Description	Value
layerRateList	Filters TPs based on layer rates while collecting in-use and potential CTPs. An empty list indicates to the element management system (EMS) to report all CTPs of all rates.	Valid values: Comma separated list of layer rates as numerical values.
managedElementManagerName	Indicates the name of the ME manager.	N/A
meFetchSize	Indicates number of MEs to fetch at a time, as opposed to obtaining them all at once.	N/A
mlsnManagerName	Name of the MultiLayerSubnetworkMgr_I manager used to obtain the manager from EmsSession_I.getManager.	N/A
modelCollectionType	Indicates whether to model logical or physical devices, or both.	Valid values: logical, physical, both (=null)
namingService	EMS naming service.	N/A
namingServiceFormat	The EMS naming service format. A value of **STRINGIFIED** indicates that the namingService property value is a CORBA stringified object reference. A value of **PLAIN** indicates that the namingService property value is in a specific format.	Valid values: PLAIN, STRINGIFIED
rootPOA	Indicates the name of the root Portable Object Adapter (POA)	N/A
topologicalLinkCollectionType	Indicates the method used to collect topological links: • A value of **BETWEEN_SN** collects topological links between subnetworks only. • A value of **INSIDE_SN** collects topological links inside subnetworks only. • A value of **ALL** collects all topological links. Topological links are not collected if value is set to **NONE**. Ensure the correct value is used, according to your vendor specifications.	Possible values: ALL, BETWEEN_SN, INSIDE_SN, NONE
tpFetchSize	Indicates the number of TPs to fetch at a time, as opposed to obtaining them all at once.	N/A
XCPipeFlushSize	Cross-connect Information Model objects are flushed to the database in batches. This value indicates number of modeled objects flushed to DB in each batch.	Valid value: integer

TMF814 Session Manager

This processor creates a session manager instance (of type oracle.communications.integrity.tmf814discovery.session.SessionManager) that is responsible for managing the EmsSession and TMF814Object managers, as well as creating and managing the emsMgr.EMSMgr_I and multiLayerSubnetwork.MultiLayerSubnetworkMgr_I managers.

This processor also populates the discovered EMS version and updates the TMF814 properties Java bean object.

TMF814 Device Recorder Initializer

This processor initializes Recording Mode (if it has been enabled). See "About Recording Mode" for more information.

TMF814 ME Collector

This processor retrieves a list of MEs using the TMF814 ME Manager. It outputs an Iterable for each ME. To deal with a large number of objects, these iterators can retrieve MEs in chunks (pagination) instead of all at one time.

Pagination is internal to the produced Iterable. The meFetchSize property set in tmf814Properties indicates the number of MEs to be retrieved at a time.

This processor can filter MEs based on name-matching criteria provided through scan parameters. Only those MEs that are matched by specified criteria are considered for further processing.

TMF814 Updated ME Discoverer

This processor retrieves a list of MEs from the NMS notifications based on the matched filter and updates **discoveryIterator** from the request with this list. It provides an output that has **notificationMENames**, **notificationManager**, and **notificationNMSDetails**. The **nmsNotificationCount** property set in **IncrementalScanParameter** provides the number of ME notifications to be retrieved.

TMF814 Device Modeler

This processor is run for each Iterable produced by the TMF814 ME Collector processor. It creates the logical and physical device entities. Device entities are not added to the result by this processor.

This processor can be configured to model either physical or logical objects by setting the modelCollectionType property. By default, both types of objects are modeled.

TMF814 Equipment Collector

This processor retrieves a list of Equipment and EquipmentHolders objects for the MEs using the Equipment Inventory Manager. It outputs an Iterable for each EquipmentOrHolder object.

TMF814 Equipment Modeler

This processor is run for each Iterable produced by the TMF814 Equipment Collector processor. It creates the equipment and equipment holder entities and adds them to the Physical Tree. This processor returns either Information Model Equipment or Equipment Holder, depending on which is modeled. See "About Cartridge Modeling" for more information.

TMF814 PTP Collector

This processor is run for each Iterable from the TMF814 Equipment Collector processor. This processor collects all the PTPs for each equipment object. It outputs an Iterable for each PTP.

TMF814 PTP Modeler

This processor is run for each Iterable from the TMF814 PTP Collector processor. This processor models each PTP as a Physical Port or Device Interface object and adds them to either the Physical or Logical Tree.

TMF814 CTP Discoverer for PTP

This processor recursively retrieves and models CTPs for each input PTP obtained from the Iterable produced by the TMF814 PTP Collector processor. The following operation is run for each PTP:

1. Using an input PTP, a TMF814 operation is run to obtain all its contained CTPs.
2. Each CTP is modeled as a Device Interface object.
3. (Optional) The CTP customizer is run.

> **Note:**
>
> The ctpModelCustomizerImplClass class is used to configure the CTP customizer. This class is set by the TMF814 Property Initializer processor.

4. The CTP is added to the Logical Tree.

Depending on the ctpCollectionDepth parameter value, a TMF814 operation is run for each collected CTP to obtain and process its child CTPs.

TMF814 FTP Collector

This processor retrieves a list of all FTPs and outputs an Iterable for each FTP object. A property set in tmf814Properties specifies whether to collect FTP details. The produced Iterable is similar to the one explained for the TMF814 ME Collector processor.

TMF814 FTP Modeler

This processor is run for each Iterable produced by the TMF814 FTP Collector processor. This processor creates Device Interface objects for the input FTPs and adds them to the Logical Tree.

TMF814 CTP Discoverer for FTP

This processor recursively retrieves and models CTPs for each input FTP obtained from the Iterable produced by the TMF814 FTP Collector processor. The following operation is run for each FTP:

1. Using an input FTP, a TMF814 operation is run to obtain all its contained CTPs.
2. Each CTP is modeled as a Device Interface object.
3. (Optional) The CTP customizer is run.

> **Note:**
>
> The ctpModelCustomizerImplClass class is used to configure the CTP customizer. This class is set by the TMF814 Property Initializer processor.

4. The CTP is added to the Logical Tree.

Depending on the ctpCollectionDepth parameter value, the above TMF814 operation is run for each collected CTP to obtain and process its child CTPs.

TMF814 Device Persister

This processor adds the logical and physical devices to the result and persists it. This processor closes and discards any CORBA iterators used.

TMF814 Device Recorder Persister

This processor persists the recorded data to a file, if the Recording Mode is enabled. See "About Recording Mode" for more information.

Update ME Notification Status

This processor uses **notificationMENames**, **notificationManager**, and **notificationNMSDetails** as input and updates the ME notifications with **PROCESSED** status.

TMF814 SNC Discoverer

This processor uses **customProperties**, **notificationManager**, **notificationNmsDetails**, and **tmf814Properties** as input. It provides **emsName**, **notificationSNCNames** and **sncList**, and lists of SNCs from the SNC notifications based on the matched filter, as output.

TMF814 Cross-Connect Discoverer

This processor collects and models cross-connects according to the following operation:

1. Run TMF814 operation to collect cross-connects.
2. For each collected cross-connect:
   a. Model the cross-connect according to the Optical Model for Network Integrity. See *Network Integrity Developer's Guide* for more information.
   b. (Optional) Run the Cross-connect Customizer processor.

> **Note:**
>
> The crossConnectModelCustomizerImplClass is used to configure the Cross-connect Customizer processor.

   c. Add modeled entity to the result group.
   d. Send last result group, or any result group equal to the configured flush size to the Network Integrity database.

Cross-connect collection is controlled by the crossConnectCollectionType parameter.

Cross-connects are modeled as pipe entities and sent to the Network Integrity database in batches. Batch sizes are configurable using the XCPipeFlushSize property.

Cross-connect modeling can be extended by creating a Cross-connect Customizer processor. See "About Design Studio Extension" for more information.

TMF814 SNC CC Discoverer

This processor uses **customProperties**, **snc**, **tpDetailMap**, **tmf814Properties**, **tmfNameToDeviceNameMap** as input. It discovers and models cross-connects for each **snc** in the for-loop. The cross-connect modeling is similar to the TMF814 Cross-Connect Discoverer processor.

Update SNC Notification Status

This processor uses **notificationMENames**, **notificationManager**, and **notificationNMSDetails** as input and updates SNC notifications with the **PROCESSED** status.

TMF814 Topological Link Collector

This processor collects all the EMS STM links and returns an Iterable that collects Topological Link objects. The produced Iterable is similar to the one explained for the TMF814 ME Collector processor.

TMF814 Updated Topological Link Collector

This processor searches topological link names from the **TopologicalLink** notifications based on filter match and collects all the EMS STM links based on the **topologicalLink** names. It returns an **Iterable** that collects the Topological Link objects. The produced Iterable is similar to the one explained for the TMF814 ME Collector processor.

TMF814 Topological Link Modeler

This processor is run for each Iterable produced by the TMF814 Topological Link Collector processor. This processor models each input topological link object according to the Optical Model for Network Integrity, and adds it to the result group. See *Network Integrity Developer's Guide* for more information.

TMF814 Pipe Persister

This processor persists all the cross-connect and topological link pipes and writes the recorded data to corresponding files. See "About Recording Mode" for more information about the recorded data files.

Update TL Notification Status

This processor uses **notificationMENames**, **notificationManager**, and **notificationNMSDetails** as input and updates the **TopologicalLink** notifications with **PROCESSED** status.

Discover TMF814 Action

This action, which extends the Discover Abstract TMF814 actions, is a complete and deployable action, configured using scan parameters, so you have full control over what is and is not discovered. This action can be extended to add new scan parameters, but the original scan parameters must remain. This action can also be extended to discover additional types of network and connectivity objects.

This discovery action inherits all the processors from the following actions:

- The Discover Abstract CORBA action

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

- The Discover Abstract TMF814 action

  For information about the inherited processors in this action, see "Discover Abstract TMF814 Action".

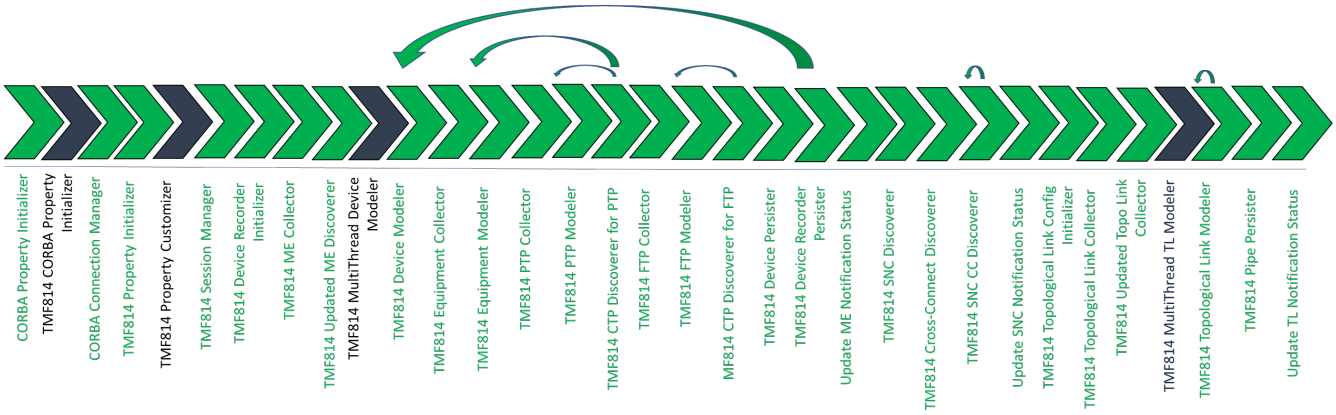
The Discover TMF814 action contains the following processors run in the following order:

1. CORBA Property Initializer (inherited)
2. TMF814 CORBA Property Initializer
3. CORBA Connection Manager (inherited)
4. TMF814 Property Initializer (inherited)
5. TMF814 Property Customizer
6. TMF814 Session Manager (inherited)
7. TMF814 Device Recorder Initializer (inherited)
8. TMF814 ME Collector (inherited)
9. TMF814 Updated ME Discoverer (inherited)
10. TMF814 MultiThread Device Modeler
11. TMF814 Device Modeler (inherited)
12. TMF814 Equipment Collector (inherited)
13. TMF814 Equipment Modeler (inherited)
14. TMF814 PTP Collector (inherited)
15. TMF814 PTP Modeler (inherited)
16. TMF814 CTP Discoverer for PTP (inherited)
17. TMF814 FTP Collector (inherited)
18. TMF814 FTP Modeler (inherited)
19. TMF814 CTP Discoverer for FTP (inherited)
20. TMF814 Device Persister (inherited)
21. TMF814 Device Recorder Persister (inherited)
22. Update ME Notification Status (inherited)
23. TMF814 SNC Discoverer (inherited)
24. TMF814 Cross-Connect Discoverer (inherited)
25. TMF814 SNC CC Discoverer (inherited)
26. Update SNC Notification Status (inherited)
27. TMF814 Topological Link Config Initializer (inherited)
28. TMF814 Topological Link Collector (inherited)
29. TMF814 Updated Topo Link Collector (inherited)
30. TMF814 MultiThread TL Modeler

31. TMF814 Topological Link Modeler (inherited)
32. TMF814 Pipe Persister (inherited)
33. Update TL Notification Status (inherited)

Figure 2-2 illustrates the processors workflow of the Discover Abstract TMF814 action.

Figure 2-2 Discover TMF814 Action Processors



TMF814 CORBA Property Initializer

This processor reads the ORBProperties and ORBArguments parameters from the UI and passes them to the CORBA Connection Manager processor through the corbaSeed. ORBProperties and ORBArguments are used during ORB initialization.

See *Network Integrity CORBA Cartridge Guide* for information about ORBProperties and ORBArguments.

TMF814 Property Customizer

This processor takes scan parameters for a specific scan and assigns them to properties in the TMF814 Property Initializer processor. See Table 7-3 for a list of available scan parameters.

TMF814 MultiThread Device Modeler

This processor models logical and physical device entities, collects and models equipments, collects and models PTPs and respective CTPs, and collects and models FTPs and the corresponding CTPs. It creates logical and physical device entities and persists to the result.

This processor verifies whether a scan is configured with the **Parallel Process** option enabled. If not enabled, the processor skips the process.

This processor uses WebLogic's **ManagedExecutorService** work-manager concept to process it in parallel. For more information, see **Working with Application Context Work-Managers** in *NI Developer's Guide*.

TMF814 MultiThread TL Modeler

This processor models topological link objects according to the Optical Model for Network Integrity, and adds it to the result group and persists the results.

This processor verifies whether a scan is configured with the **Parallel Process** option enabled. If not enabled, it skips the process.

This processor uses WebLogic's **ManagedExecutorService** work-manager concept to process it in parallel. For more information, see the **Working with Application Context Work-Managers** section in *NI Developer's Guide*.

About Recording Mode

The Optical TMF814 CORBA cartridge can be configured to record all discovered MEs, topological links, and cross-connects. 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 *WL_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 where your WebLogic domain is configured.
- *Scan_Name* is the name of the scan.

If the TMF814 scan action type has been configured to not discover MEs, topological links, or cross-connects, the corresponding file is not generated.

Recording Mode is controlled with the **tmf814.properties** file in the *WeLogic_Domain_Home***/config/corbaConfig/** directory. Recording Mode can be enabled or disabled by an administrator without needing any server or application restart. The recording processor reads this file each time it is run.

Enabling Recording Mode

To enable recording mode:

1. Open the *WebLogic_Home/***config.corbaConfig/tmf814.properties** file.
2. Search for the line: **MODE=NORMAL**
3. Change **NORMAL** to **RECORD.**
4. 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 Optical TMF814 CORBA cartridge.

Creating a Discover TMF814 Scan

The Optical TMF814 CORBA cartridge allows you to create a Discover TMF814 scan.

To create a Discover TMF814 scan:

1. Create a scan, as explained in the Network Integrity Help.
2. On the **General** tab, do the following:
   - From the **Scan Action** list, select one of the following:
     - To discover entities from a generic TMF814 element or network management system, select **Discover TMF814**.
     - To discover entities from a Huawei U2000, select **Discover Huawei U2000**.

     The **Scan Type** field displays `Discovery`.
   - Enter the following TMF814 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 depth 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. 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 7-3 for more information.

3. 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.

4. Make any other required configurations.

4

About Collected Data

This chapter explains how the Oracle Communications Network Integrity Optical TMF814 CORBA cartridge treats collected data.

About Collected Data

The Oracle Communications Network Integrity Optical TMF814 CORBA 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 infinite nesting levels. LayeredParameters are not top-level MTNM objects. They are the 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 Optical TMF814 CORBA cartridge:

- Table 4-2
- Table 4-3
- Table 4-4
- Table 4-5
- Table 4-6
- Table 4-7

Table 4-2 Managed Elements 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 allows 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 4-3 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.

Table 4-3 (Cont.) Equipment Properties

Property Name	Description
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 calling additional information and allowing 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 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
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.

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

Property Name	Description
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, but 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, but some parameter names and values may be predefined. Some predefined parameter names may include: AlarmReporting, AllocatedNumber, ASAPpointer, FragmentServerLayer, NetworkAccessDomain.

Layer Parameters

The Optical TMF814 CORBA 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 TMF814 documentation.

TMF814 APIs

This section describes the APIs used by the Optical TMF814 CORBA cartridge to collect data.

CORBA APIs

Table 4-8 lists the APIs used by the Optical TMF814 CORBA cartridge.

Table 4-8 TMF814 ManagedElement 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. • endSession(): used to close the EMS session.

Table 4-8 (Cont.) TMF814 ManagedElement and Equipment CORBA APIs

API	Used Operations
org.tmforum.mtnm.managedElementManager.ManagedElementMgr_I	• getAllFTPs(): used to obtain all FTPs, but not obtain any 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 nmsSesion while getting a Ems session so a dummy implementation is provided.
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 cross-connect collection.

Table 4-10 TMF814 Topological Link Collection APIs

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

The EMSMgs API is used when the entire network is treated as a subnetwork. The MLSN API is used when each ME is treated as a subnetwork.

Handling Vendor Variations

This section explains how the Optical TMF814 CORBA cartridge handles some of the particular data collected from some vendors.

FTP Collection API Variations

The ManagedElementMgr_I.getAllFTP() operation, from MTNM version 3.0, is the preferred API to get all FTPs of a 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 productized Optical TMF814 CORBA 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 Optical TMF814 CORBA cartridge to collect and model protection role information. See "Collecting and Modeling Protection Role Information" for more information.

5

About Cartridge Modeling

This chapter explains how the Oracle Communications Network Integrity Optical TMF814 CORBA cartridge models collected data.

About Cartridge Modeling

The Oracle Communications Network Integrity Optical TMF814 CORBA 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 since 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
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 infinite nesting) {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 Logical device acts as a container for top level interfaces. Its name is same as ME name.
PTP	DeviceInterface	tmf814TPInterfaceGeneric PTP as Interface is a container for child CTP.
FTP	DeviceInterface	tmf814TPInterfaceGeneric FTP as Interface is a container for 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
- Table 5-4
- Table 5-5
- Table 5-6
- Table 5-7
- Table 5-8
- Table 5-9

Table 5-3 Physical Device Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type	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 tmf814MEGeneric specification.	TMF814 MEGeneric

Table 5-3 (Cont.) Physical Device Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type	UI Label
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 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
nativeEmsName	Static	nativeEmsName	Text	Native EMS Name
expectedObjectType	Dynamic	expectedEquipmentObjectType	Text	Expected Object Type

Table 5-4 (Cont.) Equipment 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 corresponds to a TMF814 value: IN_SERVICE, OUT_OF_SERVICE, OUT_OF_SERVICE_BY_MAINTENANCE, SERV_NA. TMF814 does not have equivalent for TESTING.	Service State
userLabel	Dynamic	userLabel	Text	Label
owner	Dynamic	owner	Text	Owner

Table 5-5 EquipmentHolder Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type	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 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	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

Table 5-6 (Cont.) Physical Port Field Mapping

Information Model Attribute	Information Model Support	TMF Attribute	Type	UI Label
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	tpProtectionAssociation	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)
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
nativeEmsName	Static	nativeEmsName	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 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
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

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

Information Model Attribute	Information Model Support	TMF Attribute	Type and Values	UI Label
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 DeviceInterfaceConfigurationItem 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 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
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, SERV_NA. TMF814 does not have 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*"
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 Optical TMF814 CORBA cartridge models the collected cross-connect data.

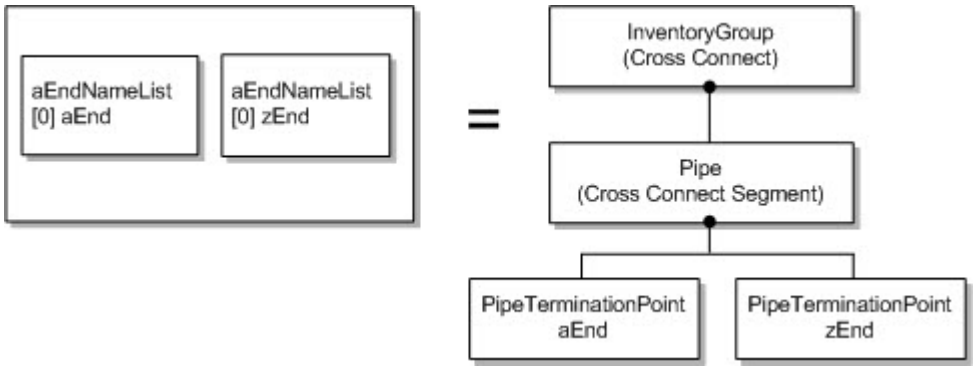
Only the cross-connect data required for assimilation is modeled. Of the data required for assimilation, only the data meeting the following conditions is modeled:

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

The Optical TMF814 CORBA cartridge models cross-connects as one of the following types:

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

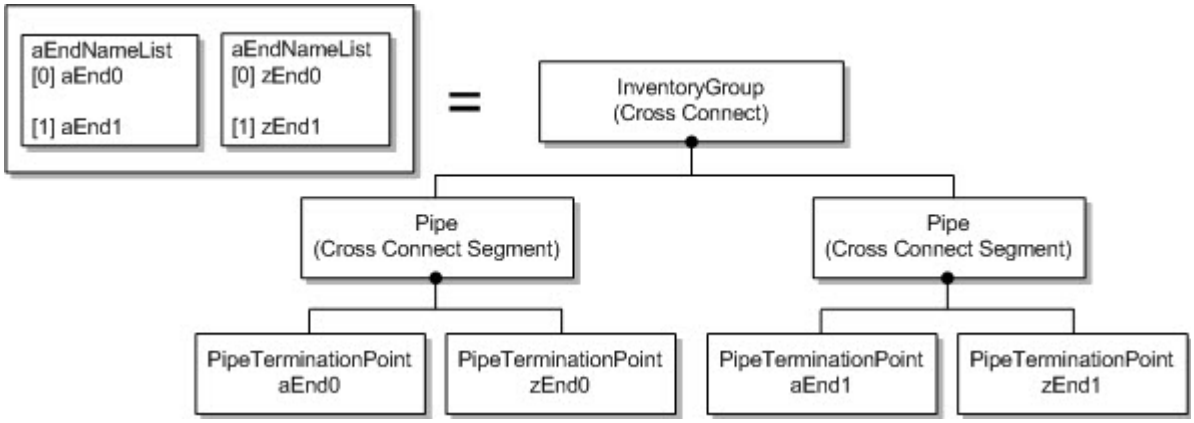
Figure 5-1 ST_SIMPLE Type Cross-Connect Model Mapping



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.

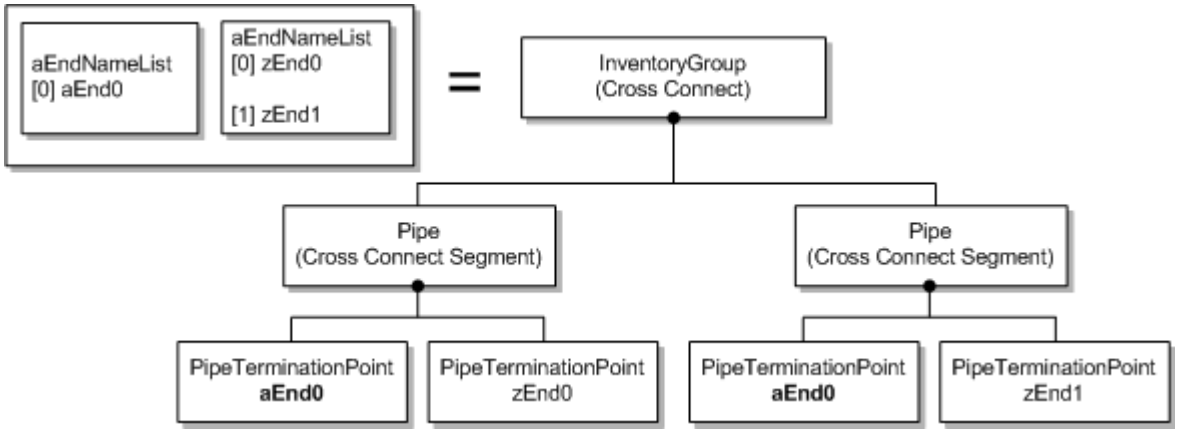
Figure 5-2 ST_EXPLICIT Type Cross-Connect Model Mapping



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.

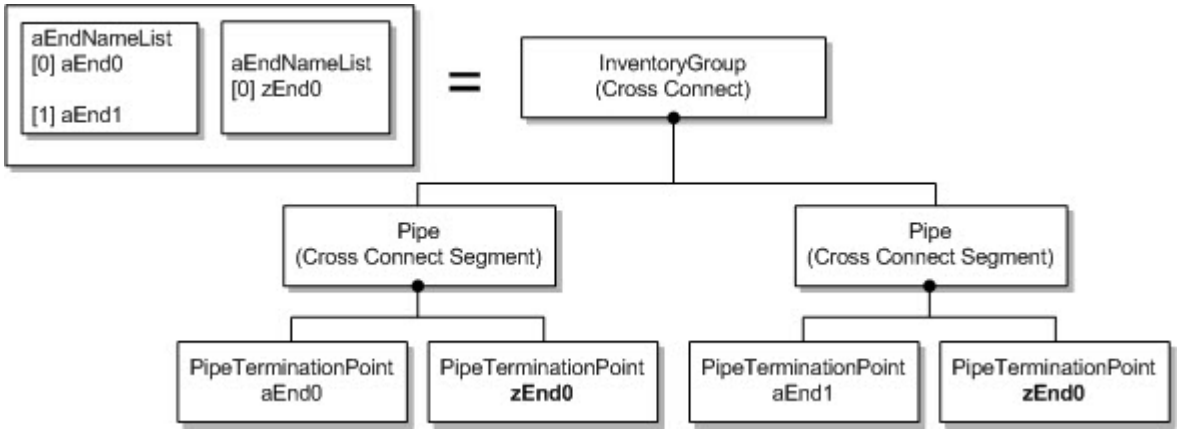
- ST_ADD_DROP_A: The cross-connect object is modeled as two pipe segments with aEndPoint repeating on both cross-connects segment, as shown in Figure 5-3.

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
- Table 5-12
- Table 5-13

Table 5-11 Model Mapping for the Inventory Group Object

Information Model Attribute	Information Model Support	TMF Attribute	Type	UI Label
name	Static	N/A	Text Value is hard-coded to **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	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	UI Label
name	Static	N/A	Text The name of the PTP (port) cross-connect endpoint.	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 Optical TMF814 CORBA cartridge can be extended to populate JKLM values that are implemented differently by some vendors. See "Customizing the JKLM Value Calculation" for more information.

Cartridge Modeling for Topological Link Data

This section explains how the Optical TMF814 CORBA 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
- Table 5-15

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

Information Model Attribute	Information Model Support	TMF Attribute	Type	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	nativeEMS Name	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	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

Table 5-15 (Cont.) Model Mapping for the PipeTerminationPoint Object for Topological Links

Information Model Attribute	Information Model Support	TMF Attribute	Type	UI Label
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 and cross-connect inventory group entities are both added to the same device result group, but in separate group containers.

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

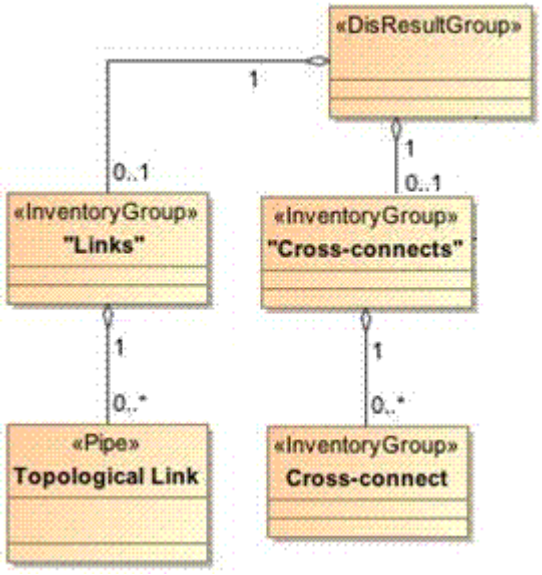
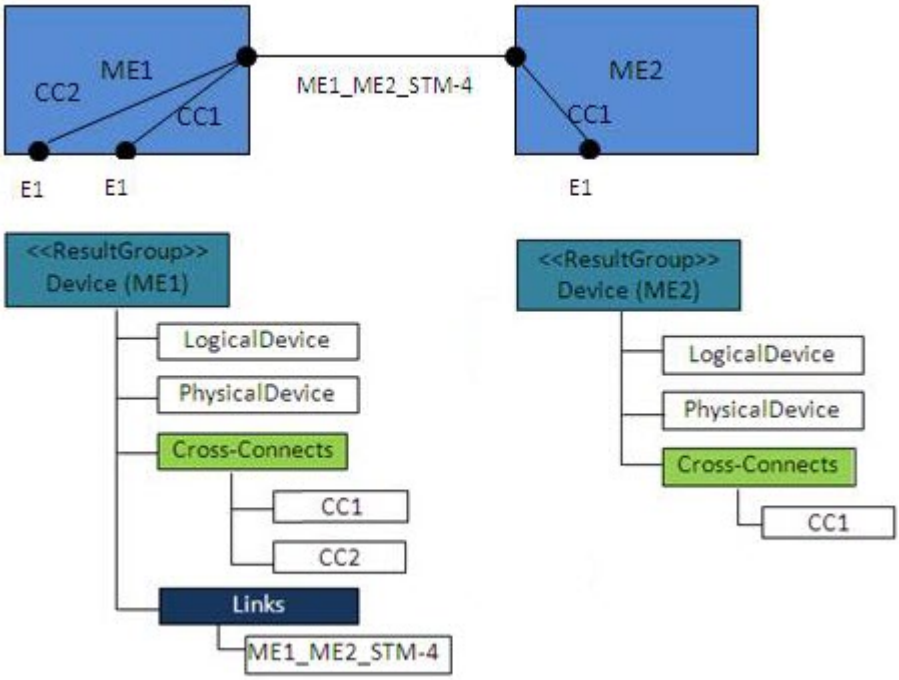


Figure 5-6 shows an example grouping for links and cross-connects with the following particularities:

- A populated result group for each device

- The appropriate cross-connects added to each device group
- The topological link is added only to the ME1 device group

Figure 5-6 Example Result Group Model and Configuration



6

About Model Correction

This chapter explains how some Multi Technology Network Management (MTNM) data is corrected to conform to Oracle Communications Information Model. Model correction occurs when the data received by the discovery action types does not conform to the Information Model. The Oracle Communications Network Integrity Optical TMF814 CORBA cartridge performs model corrections for the following:

- Equipment Holder as a Child of a Physical Device
- Sub-Slots of Slots
- Huawei U2000 MSTP End Port

Equipment Holder as a Child of a Physical Device

The MTNM model supports an Equipment Holder (rack) as a child of a physical device (managed element [ME]). The Information Model supports only Equipment as a child of physical device. Model correction is used to map a rack, shelf and sub-shelf as equipment.

Sub-Slots of Slots

When a slot has sub-slots, the sub-slots usually contain a card. The MTNM model and TMF814 consider the card as a sibling of the sub-slots. In a network, this card is the parent of sub-slots and the child of the slot. In cases where MTNM does return this sibling, model correction is used to add artificial equipment.

Huawei U2000 MSTP End Port

For Huawei U2000 MSTP devices, the physical port name of end ports is populated with the TMF name. Model correction is used to change the name to contain the native EMS name.

7

About Design Studio Construction

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

Model Collections

Table 7-1 shows the Design Studio construction of the Generic TMF814 model collection.

Table 7-1 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 Optical TMF814 CORBA cartridge actions and associated components:

- Table 7-2
- Table 7-3
- Table 7-4

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

Table 7-2 Actions Design Studio Construction

Action Name	Result Category	Address Handler	Scan Parameter Group	Processors
Discover Abstract TMF814 action	Device	N/A	IncrementalSacnParameter	• TMF814 Property Initializer • TMF814 Session Manager • TMF814 Device Recorder 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 • Update ME Notification Status • TMF814 SNC Discoverer • TMF814 Cross-Connect Discoverer • TMF814 SNC CC Discoverer • Update SNC Notification Status • TMF814 Topological Link Config Initializer • TMF814 Topological Link Collector • TMF814 Updated Topo Link Collector • TMF814 Topological Link Modeler • TMF814 Pipe Persister • Update TL Notification Status
Discover TMF814 action	Device	N/A	TMF814Parameters. See Table 7-3 AutoResolutionParameter. See *Network Integrity Developer's Guide*. Parallel Process Parameters. See *Network Integrity Developer's Guide*.	• TMF814 CORBA Property Initializer • TMF814 Property Customizer • TMF814 MultiThread Device Modeler • TMF814 MultiThread TL Modeler
Discover Huawei U2000 action	Device	N/A	Parallel Process Parameters. See *Network Integrity Developer's Guide*.	• Huawei Customizer • Huawei MSTP EndPoint Collector • Huawei MSTP EndPoint Modeler • TMF814 Huawei MultiThread Device Modeler

Table 7-3 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
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 ME. This parameter works in combination with Managed Element Name Qualifier. This parameter helps to filter the scan.	Managed Element Name
ManagedElementNameQualifier	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 NE. This parameter works in combination with Network Element Names Qualifier. This parameter helps to filter the scan.	Network Element Names
NetworkElementNameQualifier	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
CrossConnectCollectionType	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

Table 7-3 (Cont.) TMF814 Scan Parameters Design Studio Construction

Parameter Name	Parameter Type	Description	UI Label
TopologicalLinkCollection Type	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 7-4 Discovery Processor Design Studio Construction

Discovery Processors	Variable
CORBA Property Initializer	Input: N/A Output: • corbaSeed(oracle.communications.integrity.abstractcorbacartridge.CorbaSeed) A JavaBean that holds properties related to the CORBA cartridge. See *Network Integrity CORBA Cartridge Guide* for more information.
TMF814 CORBA Property Initializer	Input: corbaSeed Output: corbaSeed
CORBA Connection Manager	Input: corbaSeed Output: • namingServer(org.omg.CosNaming.NamingContextExt) • orb(org.omg.CORBA.ORB) See *Network Integrity CORBA Cartridge Guide* for more information.
TMF814 Property Initializer	Input: N/A Output: • tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties) A JavaBean that contains the set of TMF814 properties. See Table 2-1 for a list of properties. • tpDetailMap(oracle.communications.integrity.tmf814discovery.model.tp.TPNameMap) A map listing properties to PTP names.
TMF814 Property Customizer	Input: • tmf814Properties • orb • namingServer Output: tmf814Properties
Huawei Customizer	Input: • customProperties • tmfNameToDeviceNameMap (oracle.communications.integrity.tmf814discovery.model.DeviceNameMapping) Output: • tmfNameToDeviceNameMap (oracle.communications.integrity.tmf814discovery.model.DeviceNameMapping)
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.

Table 7-4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processors	Variable
TMF814 Device Recorder Initializer	Input: tmf814Properties, customProperties Output: • recordMode(boolean) A Boolean indicating whether Recording Mode is enabled.
TMF814 ME Collector	Input: N/A Output: • meIterable Iterable object for each collected ME.
TMF814 Device Modeler	Input: • tmf814Properties • customProperties • managedElement(org.tmforum.mtnm.managedElement.ManagedElement_T) One instance of the meIterable. 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.
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. This processor is run once for each instance of equipmentOrHolder. Output: • equipment(oracle.communications.inventory.api.entity.Equipment) Returned value if input is equipment. • equipmentHolder(oracle.communications.inventory.api.entity.EquipmentHolder) Returned value if input is equipment holder.

Table 7-4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processors	Variable
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.
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.

Table 7-4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processors	Variable
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
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
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 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.

Table 7-4 (Cont.) Discovery Processor Design Studio Construction

Discovery Processors	Variable
TMF814 Pipe Persister	Input: tmf814Properties, tpDetailMap, tlPipeMap Output: n/a

8

About Design Studio Extension

This chapter contains examples and explanations on how to extend certain aspects of the Oracle Communications Network Integrity Optical TMF814 CORBA cartridge 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

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 Oracle Communications 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.

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

Extending the Discover TMF814 Action to Collect Vendor-Specific Information

This example explains how to model 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 Optical TMF814 CORBA 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 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 helper calls 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 8-1.

Figure 8-1 Custom Device Modeler Processor Workflow

Collecting Vendor-Specific Details for CTPs

This example explains how to model vendor-specific details about connection termination points (CTPs). CTP collection is handled differently from other objects because both the collecting and the modeling are handled by the same processor. In this example, vendorState of a CTP is used as the sought 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 Optical TMF814 CORBA 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. In the Studio Projects view, expand the **TMF814_Model** project.
7. Expand the Device Interface specifications and copy the tmf814TPInterfaceGeneric specification.
8. Copy the Device Interface into your project and call it customTMF814TPInterfaceGeneric.
9. Add a characteristic named vendorState to the customTMF814TPInterfaceGeneric specification.
10. Change to the Java perspective.
11. In the TMF814SampleVendorExtension project, add a class that implements the oracle.communications.integrity.tmf814discovery.model.ctp.CTPModelCustomizer interface, as shown in the following example:

```
/**
* This is a CTP model customizer.
*/

package com.vendor.ctp;
import oracle.communications.integrity.tmf814discovery.model.ctp.CTPModelCustomizer;
import oracle.communications.integrity.tmf814discovery.model.ocimtree.LogicalTree;
import oracle.communications.inventory.api.entity.DeviceInterface;
import org.tmforum.mtnm.terminationPoint.TerminationPoint_T;

public class CTPModelCustomizerImpl implements CTPModelCustomizer {

  /**
  * Overriding customize method.
  * @param inter, modeled DeviceInterface
  * @param tp, TerminationPoint_T tmf object
  * @param tree
  */

  public void customize(DeviceInterface inter, TerminationPoint_T tp,
LogicalTree<Object> tree) {
    //1. Get vendor data from termination point.
    //2. Create new specification
    //3. Set vendor specific data to Information Model data.
  }
}
```

12. Switch back to the Design Studio perspective.
13. Register the CTPModelCustomizerImpl class:
    a. Add a new discovery processor to the TMF814 Sample Vendor Extension action. This processor should be added after TMF814 Property Customizer. Name the processor Vendor Property Customizer.
    b. For the Vendor Property Customizer processor, set the following context parameters:
       - Input: tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties)
       - Output: tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties)
    c. Create a Java implementation for the Vendor Property Customizer processor by adding code similar to the following example to the processor implementation:

```
import oracle.communications.integrity.scanCartridges.sdk.ProcessorException;
import
oracle.communications.integrity.scanCartridges.sdk.context.DiscoveryProcessorCont
ext;
import oracle.communications.integrity.tmf814discovery.beans.TMF814Properties;

public class VendorPropertyCustomizerProcessorImpl implements
VendorPropertyCustomizerProcessorInterface {

  @Override
  public VendorPropertyCustomizerProcessorResponse invoke(
  DiscoveryProcessorContext context,
  VendorPropertyCustomizerProcessorRequest request) throws ProcessorException {
    //Get properties from request
    TMF814Properties prop = request.getTmf814Properties();
```

```
    //Set fully qualified name of above CTP customizer implementation class.
    prop.setCtpModelCustomizerImplClass("com.vendor.ctp.CT
PModelCustomizerImpl");
    //Create processor response and set the prop to the response
    VendorPropertyCustomizerProcessorResponse response = new
VendorPropertyCustomizerProcessorResponse();
    response.setTmf814Properties(prop);
    return response;
  }
}
```

14. Save and close all files.
15. Build, deploy, and test your cartridge.

Adding New Managers

To add a new manager:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the Optical TMF814 CORBA cartridge project.
4. Create a discovery action named TMF814 New Manager Extension.
5. Add the Discover TMF814 action as a processor in the TMF814 New Manager Extension action.
6. Create a new processor named My Manager Initializer and insert it after the TMF814 Session Manager processor.
7. For the My Manager Initializer processor, set the following context parameters:
   - input: sessionManager(oracle.communications.integrity.tmf814discovery.session.SessionManager)
   - output: all managers initialized by the processor
8. Add the necessary logic to initialize the managers in the generated MyManagerInitilizerImpl class invoke() method.

   The following example shows the logic to initialize a new manager named My Manager:

   ```
   oracle.communications.integrity.tmf814discovery.session.SessionManager
   sessionManager = request.getSessionManager();

   //This method in sessionManager tries to create a manager identified by //specified
   manager name using currently active Ems Session.

   org.omg.CORBA.Object obj = sessionManager.getManager("My_Manager_Name");

   //Narrow the generic CORBA object to specific type.
   My_Manager myManaer = My_Manager_IHelper.narrow(obj);
   ```

9. Add all the managers to the response object of the invoke() method.

   All the processors following the TMF814 Manager Initializer processor can make use of the newly initialized managers.

> **Note:**
>
> If the new managers are used to collect new objects, you should design corresponding collector, discoverer, and modeler processors.
>
> If you wish to record the results, the discoverer processor needs to extend the TMF814 Device Recorded Initializer and TMF814 Device Recorder Persister processors. The postProcess() method needs to be run after each object is fetched. Update the writeRecord() method Java class for any new objects.

Creating a Custom Equipment Reconciliation Cartridge

You can create a custom equipment reconciliation cartridge that discovers your TMF814 equipment and reconciles it with your inventory system.

To create a custom equipment reconciliation cartridge:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the Optical TMF814 CORBA cartridge project.
4. Create a discovery action named TMF814 Equipment Reconciliation.
5. Add either the Discover TMF814 action Discover Abstract TMF814 action as a processor in the TMF814 Equipment Reconciliation action.
6. Create an import action to import your inventory data into Network Integrity.
7. Create a discrepancy detection action to compare your discovered TMF814 equipment data with your imported inventory equipment data.
8. Create a discrepancy resolution action to correct discrepancies between your discovered TMF814 equipment data and your imported inventory equipment data.
9. Build, deploy, and test your cartridge.

See *Network Integrity Optical UIM Integration Cartridge Guide* for more information if you use Oracle Communications Unified Inventory Management (UIM) as your inventory system.

Creating a Custom Circuit Reconciliation Cartridge

You can create a custom circuit reconciliation cartridge that discovers your TMF814 circuit data and reconciles it with your inventory system.

Some element management systems (EMSs) and network management systems (NMSs) use a device model where the connection termination point (CTP) circuit name is assigned to the userLabel attribute. The following example requires that the EMS or NMS device model use the userLabel attribute of a CTP to hold the circuit name.

To create a custom circuit reconciliation cartridge:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Create an import action that does the following:
   - Retrieves physical devices

- Retrieves logical devices
- Maps circuit channel assignments by setting the userLabel attribute of the channel subinterface to the circuit name

4. Create a discrepancy detection action.
5. (Optional) Extend the discrepancy detection action to allow Network Integrity to search and group the discrepancy results by doing the following:
   a. Extend the initializer processor to register a filter against the device interface.
   b. Extend the action to run the filter when a discrepancy is detected against an interface. Populate either the **Priority** or **Owner** field with the corresponding circuit name.

   > **Tip:**
   >
   > In the event of an attribute mismatch discrepancy, obtain the circuit name from the compareEntity entity.
   >
   > In the event of a missing or extra entity, obtain the circuit name from the childTargetEntity entity.

   This action obtains the circuit name and adds it to either the **Owner** or **Priority** field in the Network Integrity UI, allowing you to search for and sort by the circuit name.
6. Create a discrepancy resolution action to handle circuit discrepancies.

   > **Note:**
   >
   > A channel assignment discrepancy may exist due to an incorrect userLabel attribute on the subinterface, or an extra or missing entity on the subinterface.
   >
   > If the circuit exists in the network but is missing from the inventory system, the discrepancy detection action returns multiple discrepancies. The reconciliation action may need to perform additional operations to correct a missing circuit.
7. Build, deploy, and test your cartridge.

Customizing the JKLM Value Calculation

You can customize the JKLM value calculation used to model collected cross-connect data.

To customize the JKLM value calculation:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the Optical TMF814 CORBA cartridge project.
4. Create a discovery action that uses the Discovery TMF814 action as a processor.
5. Create a new processor called XCModelCustomizer and insert it after the TMF814 Property Customizer processor.
6. For the XCModelCustomizer processor, set the following context parameters:

- input:
  tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties)

7. Develop a new MyXCCustomizer.java Java class by implementing the oracle.communications.integrity.tmf814discovery.model.xc.XCModelCustomizer class with the customize() method.
8. Develop the customize() method of the MyXCCustomizer.java class to customize the JKLM values for each cross-connect type.
9. Develop the invoke() method of the MyXCCustomizer.java class to set the new class in the setXCModelCustomizerImplClass class, as shown in the following example:

```
request.getTmf814Properties().setXCModelCustomizerImplClass("MyXCCustomizer")
```

10. Build, deploy and test your cartridge.

    The XCModelCustomizer processor is run in the order shown by Figure 8-2. All the processors following the XCModelCustomizer processor can make use of the newly initialized managers.

Figure 8-2 Customized JKLM Value Calculation Processors Workflow

Adding New CORBA API Calls

This example explains how arbitrary TMF814 objects that are not collected by default by the Optical TMF814 CORBA cartridge can be collection and modeled.

This example shows the collection and modeling of the GTP_T object, though this is the general approach for any object.

To add new CORBA API calls:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the Optical TMF814 CORBA cartridge project.
4. Create a discovery action with name Discover Custom TMF814 Objects.
5. Add the Discover TMF814 action as a processor in the Discover Custom TMF814 Objects action.
6. Create a new discovery processor named TMF814 GTP Collector and insert it before the TMF814 Device Persister processor.
7. Configure the TMF814 GTP Collector processor to have the following input parameters:
   - managedElement(org.tmforum.mtnm.managedElement.ManagedElement_T)
   - sessionManager(oracle.communications.integrity.tmf814discovery.session.SessionManager)
   - tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties)
8. Configure the TMF814 GTP Collector processor to have the following output parameters:
   - gtpIterable(oracle.communications.integrity.tmf814discovery.collection.

     TMF814DiscoveryIterable< org.tmforum.mtnm.terminationPoint.GTP_T >)
9. Create the TMF814GTPCollectorProcessorImpl Java class to resemble the following example:

```
import oracle.communications.integrity.scanCartridges.sdk.ProcessorException;
import
oracle.communications.integrity.scanCartridges.sdk.context.DiscoveryProcessorContext;
import oracle.communications.integrity.tmf814discovery.beans.CustomProperties;
import oracle.communications.integrity.tmf814discovery.beans.TMF814Properties;
import
oracle.communications.integrity.tmf814discovery.collection.TMF814DiscoveryIterable;
import oracle.communications.integrity.tmf814discovery.discoverers.DiscovererRequest;
import oracle.communications.integrity.tmf814discovery.discoverers.factory.Type;
import oracle.communications.integrity.tmf814discovery.session.SessionManager;
import oracle.communications.sce.integrity.sdk.processor.ProcessorFinalizer;

import org.tmforum.mtnm.globaldefs.NameAndStringValue_T;
import org.tmforum.mtnm.terminationPoint.GTP_T;

import com.oracle.tmf814discovery.discoverers.GTPDiscoverer;

public class TMF814GTPCollectorProcessorImpl implements
TMF814GTPCollectorProcessorInterface, ProcessorFinalizer {
  private GTPDiscoverer discoverer;

  @Override
  public TMF814GTPCollectorProcessorResponse invoke(DiscoveryProcessorContext
context, TMF814GTPCollectorProcessorRequest request) throws ProcessorException {

    //Get SessionManager from request
    SessionManager mgr = request.getSessionManager();
    NameAndStringValue_T[] meName = request.getManagedElement().name;
    TMF814Properties prop = request.getTmf814Properties();

    //Create GTPDiscoverer, this has the API calls to get GTP objects from Ems
sytem.
    discoverer =  new GTPDiscoverer(meName, prop, mgr);
```

```
    DiscovererRequest req= new DiscovererRequest(request.getTmf814Properties(), new
CustomProperties());
    TMF814DiscoveryIterable<GTP_T> gtpIterable = new
TMF814DiscoveryIterable<GTP_T>(Type.OTHER,req,  discoverer);

    //Create iterable for GTP and set to resposne
    TMF814GTPCollectorProcessorResponse res = new
TMF814GTPCollectorProcessorResponse();
    res.setGtpIterable(gtpIterable);
    return res;
  }
  @Override
  public void close(boolean arg0) {
    if(discoverer != null){
      discoverer.destroy();
    }
  }
}

package com.oracle.tmf814discovery.discoverers;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import oracle.communications.integrity.scanCartridges.sdk.ProcessorException;
import oracle.communications.integrity.tmf814discovery.beans.TMF814Properties;
import oracle.communications.integrity.tmf814discovery.discoverers.TMF814Discoverer;
import oracle.communications.integrity.tmf814discovery.session.SessionManager;

import org.tmforum.mtnm.globaldefs.NameAndStringValue_T;
import org.tmforum.mtnm.globaldefs.ProcessingFailureException;
import org.tmforum.mtnm.managedElementManager.ManagedElementMgr_I;
import org.tmforum.mtnm.terminationPoint.GTP_T;
import org.tmforum.mtnm.terminationPoint.GTPiterator_I;
import org.tmforum.mtnm.terminationPoint.GTPiterator_IHolder;
import org.tmforum.mtnm.terminationPoint.GTPlist_THolder;

public class GTPDiscoverer implements TMF814Discoverer<GTP_T>{
  protected static final Logger logger =
Logger.getLogger(GTPDiscoverer.class.getName());

  private int fetchSize;
  private boolean isInitialTMFOperInvoked = false;
  private TMF814Properties tmf814Properties;
  private boolean isEOD = false; //Is End Of Discovery reached.

  private NameAndStringValue_T[] meName = null; //parent ME name
  private GTPiterator_I gtpIter = null;
  private ManagedElementMgr_I meMgr;

  public GTPDiscoverer(NameAndStringValue_T[] meName, TMF814Properties prop,
SessionManager sessionMgr) throws ProcessorException {
    this.tmf814Properties = prop;
    this.meName = meName;
    this.meMgr = sessionMgr.getManagedElementMgr();
    this.fetchSize = 1000;
  }
```

```
  /**
  *Every time the discover() method is called, certain number of objects
  *are called. Number of object to return is up the implementation.
  *Empty iterator is returned indicating that there are no objects
  *to discover/retrieve further.
  */

  @Override
  public java.util.Iterator<GTP_T> discover(){
    if(isEOD){
      return Collections.<GTP_T>emptyList().iterator();
    }

    List<GTP_T> result;
    if(! isInitialTMFOperInvoked ) {
      //Initialize GTP iteraror
      result = fetchInitialElements();
      isInitialTMFOperInvoked = true;
    }else{
      //Once GTP iteraror is initialized, fetchMoreElements is called by the
iterable.
      result = fetchMoreElements();
    }
    if(result.isEmpty() || (getFetchSize() >  result.size()) ){
      isEOD= true;
      destroy();
    }
    return result.iterator();
  }

  private int getFetchSize() {
    return fetchSize;
  }

  public List<GTP_T> fetchInitialElements() {
    GTPlist_THolder gtpListHolder = new GTPlist_THolder();
    GTPiterator_IHolder gtpIterHolder = new GTPiterator_IHolder();
    try {
      meMgr.getAllGTPs(meName, new short[]{}, 2000, gtpListHolder, gtpIterHolder);
      gtpIter = gtpIterHolder.value;
      if ( gtpListHolder.value != null) {
        return Arrays.<GTP_T>asList(gtpListHolder.value);
      }
    } catch (Exception e) {
      logger.log(Level.SEVERE, "getAllGTP: Error while getting initial gtps", e);
    }
    return Collections.<GTP_T>emptyList();
  }

  public List<GTP_T> fetchMoreElements() {
    if(gtpIter != null){
      GTPlist_THolder gtpListHolder = new GTPlist_THolder();
      try {
        gtpIter.next_n(2000, gtpListHolder);
      } catch (ProcessingFailureException e) {
        logger.log(Level.SEVERE, "getAllGTP(next_n): Error while getting more gtps",
e);
      }
      if(gtpListHolder.value != null && gtpListHolder.value.length > 0){
        return Arrays.<GTP_T>asList(gtpListHolder.value);
      }
    }
```

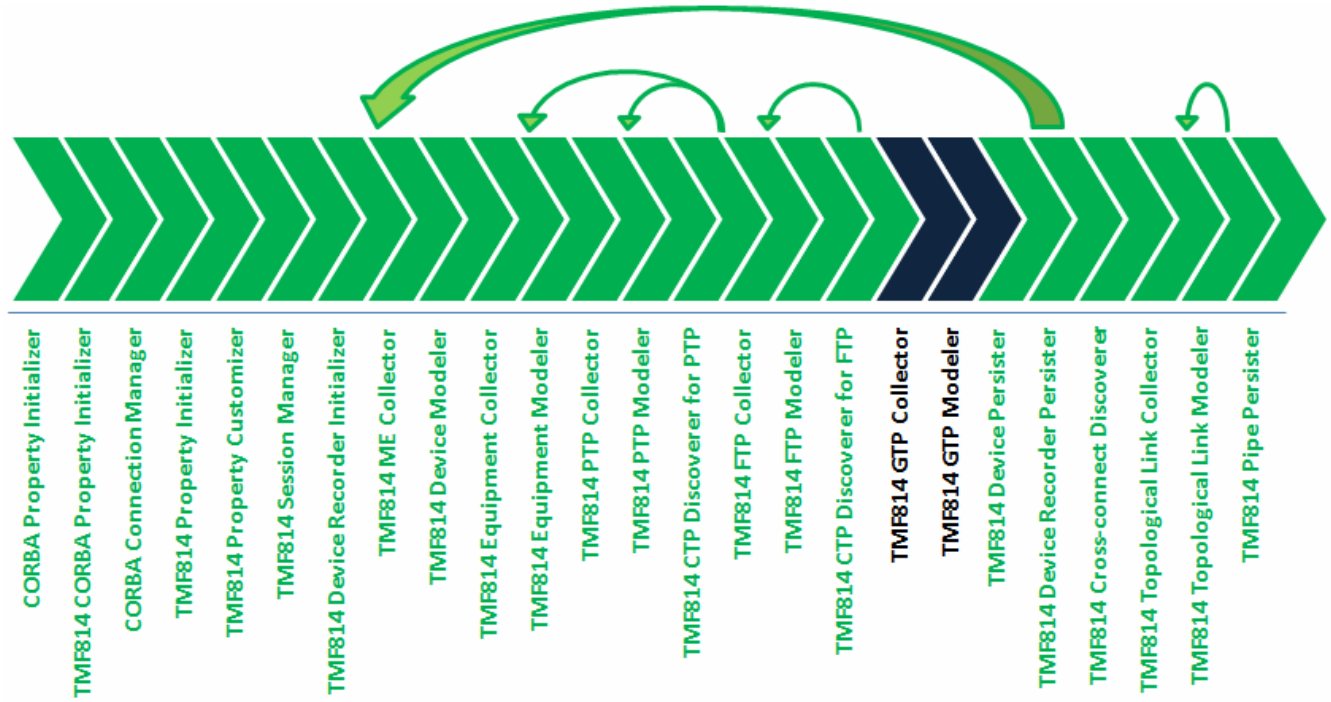
```
    return Collections.<GTP_T>emptyList();
  }

  @Override
  public void destroy() {
    if (gtpIter != null) {
      try{
        gtpIter.destroy();
      }catch (ProcessingFailureException e) {
        logger.log(Level.INFO, "exception while closing gtp iterator", e);
      }
      gtpIter = null;
    }
  }
}
```

10. Create a For Each processor after the TMF814 GTP Collector processor.
11. Specify the following values for the For Each processor:
    - In the **Select Collection Name** field, enter **gtpIterable**.
    - In the **Variable Name** field, enter **gtp**.
12. Within the For Each processor, create a processor named TMF814 GTP Modeler. The TMF814 GTP Modeler processor is responsible for modeling each input GTP_T as an Oracle Communications Information Model object and adding it to the result.
13. Configure the TMF814 GTP Modeler processor to have the following input parameters:
    - gtp(org.tmforum.mtnm.terminationPoint.GTP_T)
14. Configure the TMF814 GTP Modeler processor to have the following output parameters:
    - Modeled Information Model representation of gtp.
15. Design the TMF814 GTP Modeler processor to find the correct Information Model mapping object for the TMF814 GTP object and add to the result.
16. Build, deploy, and test your cartridge.

    The TMF814 GTP Collector and TMF814 GTP Modeler processors are run in the order shown by Figure 8-3.

Figure 8-3 New CORBA API Calls Processor Workflow



Collecting and Modeling Protection Role Information

You can extend the Optical TMF814 CORBA cartridge to collect protection role information on cross-connect segments. The protection role status can be made available to other cartridges and follow-on actions.

Because there are no known APIs to obtain protection data from devices, this scenario assumes that the protection role information is available from another source of data, such as in a CVS file.

To collect protection role data on cross-connect segments:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the Optical TMF814 CORBA cartridge project.
4. Create a discovery action that uses the Discover TMF814 action as a processor.
5. Create a discovery processor called XCModelCustomizer and insert it after the TMF814 Property Customizer processor.
6. Configure the new processor to have the following input parameter:
   - tmf814Properties(oracle.communications.integrity.tmf814discovery.beans.TMF814Properties)
7. Develop a new MyXCCustomizer.java Java class by implementing the oracle.communications.integrity.tmf814discovery.model.xc.XCModelCustomizer class with the customize() method.
8. Develop the customize() method of the MyXCCustomizer.java class to populate the protection role for each cross-connect segment, as shown in the example below:

```
import oracle.communications.integrity.tmf814discovery.model.xc.XCModelCustomizer;
import oracle.communications.inventory.api.entity.InvGroupRef;
import oracle.communications.inventory.api.entity.InventoryGroup;
import oracle.communications.inventory.api.entity.Pipe;

import org.tmforum.mtnm.subnetworkConnection.CrossConnect_T;

public class MyXCCustomizer implements XCModelCustomizer {
  public static final String PROTECTIONROLE = "protectionRole";
  @Override
  public InventoryGroup customize(CrossConnect_T xc, InventoryGroup ccGroup) {
    Set<InvGroupRef>  segmentRelSet = ccGroup.getMembers();
    for(InvGroupRef ref : segmentRelSet){
      Pipe pipe = ref.getPipe();
    //Get protection information for this segment from a data source.
      String prorectionRole = /*Get it from external source*/
      pipe.getCharacteristicMap().get(PROTECTIONROLE).setValue(prorectionRole);
    }
    return ccGroup;
  }
}
```

9. Develop the invoke() method of the MyXCCustomizer.java class to set the new class in the setXCModelCustomizerImplClass class, as in the following example:

```
request.getTmf814Properties().setXCModelCustomizerImplClass("MyXCCustomizer")
```

10. Build, deploy and test your cartridge.

    All the processors following the XCModelCustomizer processor can make use of the information.

Discovering Custom Device or Result Group Names

You can customize the way discovered devices and result groups are named to match how they are named in your inventory system.

To customize how discovered devices and result groups are named:

1. Open Design Studio in the Design perspective.
2. Create a Network Integrity cartridge project.
3. Make the cartridge project dependent on the Optical TMF814 CORBA cartridge project.
4. Create a new discovery action that uses the Discover TMF814 action as a processor.
5. Create a discovery processor and insert it after the TMF814 Property Initializer processor.
6. Set the new processor to use tmfNameToDeviceMap, the output from the TMF814 Property Initializer processor, as its input.
7. Map each device or result group to tmfNameToDeviceMap, as in the following example:

   ```
   nameToNativeEmsMap1.addMapping(tmf814_Name, custom_Name)
   ```

   Where *tmf814_Name* is the ManagedElement tuple value of ManagedElement_T.name and *custom_Name* is the custom name of the device.
8. Build, deploy and test your cartridge.