# Oracle® Communications Network Integrity

TL1 Cartridge Guide

Release 7.3.2

**E66048-01**

May 2016

ORACLE®

Oracle Communications Network Integrity TL1 Cartridge Guide, Release 7.3.2

E66048-01

# Contents

## 3 About the Cisco ONS 15454 TL1 Reference Cartridge

# Preface

This guide explains the functionality and design of the Oracle Communications Network Integrity TL1 cartridge and the design of the Cisco ONS 15454 SONET TL1 reference 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 an understanding of cartridge deployment, cartridge development, working with cartridges, working with actions, and the extensibility SDK.
- *Network Integrity Concepts*: For an understanding of the Network Integrity architecture and using Oracle Communications Design Studio for Network Integrity.

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

- Oracle Communications Design Studio and its terminology
- Oracle Communications Information Model
- Transaction Language 1 (TL1) standards and terminology
- Cisco ONS 15454 SONET TL1 commands

## Documentation Accessibility

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

### Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

## Document Revision History

The following table lists the revision history for this guide:

| Version | Date | Description |
|---|---|---|
| E66048-01 | May 2016 | Initial release. |

# 1

# Overview

This chapter provides an overview of the Oracle Communications Network Integrity TL1 cartridge and the Cisco ONS 15454 SONET TL1 reference cartridge.

## TL1 Cartridge Overview

The TL1 cartridge allows you to build deployable cartridges that interact with transaction language 1 (TL1) devices or systems.

The TL1 cartridge provides the following key features:

- Transmission control protocol (TCP), Telnet protocol, and secure shell (SSH) communication with TL1 enabled devices, gateways, and element management systems (EMSs)
- Gateway discovery
- Extensible command dictionary in Oracle Communications Design Studio for defining TL1 requests and responses
- Extensible discovery results model using Design Studio
- Record and playback of TL1 communication

The TL1 cartridge is an abstract cartridge, meaning that Design Studio is used to configure and assemble the run time cartridge against target systems or devices before deploying it into Network Integrity. Because there are no globally common commands among all TL1 devices or agents, you must build a specific extension to the TL1 cartridge to communicate to specific device or agent families. See *Network Integrity Concepts* for guidelines and best practices for extending cartridges.

The TL1 cartridge uses the WebNMS TL1 library (a set of Java-based APIs) to provide complete support for building TL1 management applications.

The TL1 cartridge ZIP file contains a reference implementation cartridge for discovering Cisco ONS 15454 SONET TL1 devices. See "Cisco 15454 TL1 Reference Cartridge Overview" for more information.

The TL1 cartridge connects to TL1 agents using TCP, Telnet, or SSH. Use SSH over TCP and Telnet as much as possible for security reasons. Reserve TCP and Telnet communication for devices that do not support SSH. The TL1 cartridge uses a Java secure channel (JSCH) library (a set of Java-based APIs) to provide SSH support.

The TL1 cartridge formats and parses TL1 requests and responses with the command dictionary. The command dictionary is a Design Studio cartridge development framework for defining command requests and responses. The command dictionary provides utilities that convert populated request types into command strings and convert device responses into structured data holders. For example, Design Studio is

used to map the TL1 responses to the Oracle Communications Information Model to support discrepancy detection and resolution.

The TL1 cartridge does not provide Discrepancy Detection or Discrepancy Resolution actions. However, the discovery results from the TL1 cartridge can be used as input for a Discrepancy Detection or Discrepancy Resolution action run by another cartridge.

See "About the TL1 Cartridge" for more information about the TL1 cartridge and its components.

# Cisco 15454 TL1 Reference Cartridge Overview

The Cisco 15454 TL1 reference cartridge shows you how to develop a cartridge that extends the TL1 cartridge to support the discovery of a specific TL1 device, gateway, or EMS. Use the Cisco 15454 TL1 reference cartridge as a model to develop your own discovery cartridge that extends the TL1 cartridge.

The Cisco 15454 TL1 reference cartridge extends the TL1 cartridge to support the collection and modeling of physical equipment data from a Cisco ONS 15454 SONET device using the TL1 protocol.

See "About the Cisco ONS 15454 TL1 Reference Cartridge" for more information about the Cisco 15454 TL1 reference cartridge and its components.

# About the Cartridge Dependencies

This section provides information about dependencies that the TL1 cartridge and Cisco 15454 TL1 reference cartridge have on other entities.

## Run-Time Dependencies

The TL1 cartridge and Cisco 15454 TL1 reference cartridge require that the Address_Handlers cartridge be deployed to Network Integrity.

## Design-Time Dependencies

The TL1 cartridge has the following dependencies:

- Address_Handlers
- Network Integrity SDK
- ora_uim_model

The Cisco 15454 TL1 reference cartridge has the following dependencies:

- Address_Handler
- NetworkIntegritySDK
- ora_uim_model
- TL1 cartridge

# Downloading and Opening the Cartridge Files in Design Studio

To review and extend the TL1 cartridge and the Cisco 15454 TL1 reference cartridge, download the TL1 cartridge ZIP file from the Oracle software delivery web site:

`https://edelivery.oracle.com`

The TL1 cartridge ZIP file has the following structure:

- Cisco_TL1_Cartridge
- TL1_Cartridge
- Address_Handlers

The **Cisco_TL1_Cartridge** project and the **TL1_Cartridge** project contain the extendable Design Studio files.

See *Network Integrity Developer's Guide* for information about opening files in Design Studio.

## Building and Deploying the Cartridge

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

# 2

# About the TL1 Cartridge

This chapter describes the Oracle Communications Network Integrity TL1 cartridge and its components.

## About Actions and Processors

The TL1 cartridge contains the following actions:

- Discover Abstract Base TL1
- Discover Abstract TL1

The Discover Abstract TL1 action extends the Discover Abstract Base TL1 action, pulling in all the functionality of that action to establish and manage the TL1 connection.

The Discover Abstract Base TL1 action contains the following processors run in the following order:

1. TL1 Property Initializer
2. TL1 Connection Manager

Figure 2–1 illustrates the processor workflow of the Discover Abstract Base TL1 action.

***Figure 2–1 Discover Abstract Base TL1 Action Processor Workflow***


The Discover Abstract TL1 action adds scan parameter groups to the Discover Abstract Base TL1 action, passing the property values to the TL1Properties object.

The Discover Abstract TL1 action contains the following processors run in the following order:

1. TL1 Property Initializer (inherited)
2. TL1 Property Customizer
3. TL1 Connection Manager (inherited)

Figure 2–2 illustrates the processor workflow of the Discover Abstract TL1 action.

***Figure 2–2 Discover Abstract TL1 Action Processor Workflow***


## TL1 Property Initializer

This processor is part of the Discover Abstract Base TL1 action.

This processor initializes all the properties required for connecting to TL1 and puts all the properties into a TL1Properties object (a Java class).

Table 2–1 lists the values that are initialized by the TL1 Property Initializer processor.

***Table 2–1 TL1 Properties Initialized by the TL1 Property Initializer Processor***

| Parameter | Description |
|---|---|
| host | Host name or IP address of the TL1 agent. |
| port | Port number of the TL1 agent. |
| userid | The user identification name. It may be from 1 to 10 characters in length. Input is case-sensitive and can consist of numeric, upper- and lower-case alphabetic characters. |
| password | The user password or private identifier. It may be up to 10 characters in length. |
| sshUser | The user ID used to establish the SSH or Telnet session. |
| sshPassword | The password used to establish the SSH or Telnet session. |
| knownHostFileLocation | The location where the public keys of the TL1 server are stored if SSH is used. Leave blank to accept all public keys. If a directory is provided, the Network Integrity server must have permission to write to it, otherwise the TL1 cartridge defaults to accepting all public keys. |
| timeout | The timeout of the underlying socket connection in seconds. |
| retries | The number of retries that are attempted after a timeout. Obtained from the TL1 scan parameter groups. |
| loginTargetIdentifier | Identifies the routing information necessary for a login message sent by an OS to reach a network element (NE). Specifying this parameter is optional. |

*Table 2–1 (Cont.) TL1 Properties Initialized by the TL1 Property Initializer Processor*

| Parameter | Description |
|---|---|
| loginCorrelationTag | Used to correlate the login input command with a response. The default is **empty**. Specifying this parameter is optional. |
| transportProtocol | Identifies the TL1 protocol to use: TCP, SSH, TELNET, or OTHER. The default is SSH. |
| customTransport | An implementation of the WebNMS TL1TransportProvider interface. If customTransport is specified, transportProtocol must be set to **OTHER**. The default is **empty**. Specifying this parameter is optional. |
| connectionCustomizer | An implementation of TL1Connection.Customizer that can be configured to handle non-standard behavior, such as a Telnet banner that is not recognized by the WebNMS TL1 API. |
| gatewayDiscovery | A Boolean value indicating whether the discovery target is a gateway. The default is **false**. |
| gatewayCommand | The command to retrieve the list of NEs from the gateway. Specifying this parameter is optional. |
| gatewayNEFilter | A filter on the NE ID returned from the TL1 gateway. Specifying this parameter is optional. |
| gatewayNEFilterQualifier | This property works in combination with GatewayNEFilter to match the collected NEs by name and qualifier. Possible values are EQUALS, EQUALS_IGNORE_CASE, CONTAINS, CONTAINS_IGNORE_CASE, STARTS_WITH, STARTS_WITH_IGNORE_CASE, ENDS_WITH, ENDS_WITH_IGNORE_CASE. Specifying this parameter is optional. |
| gatewayNEIDResponseSectionPosition | Identifies where the NE ID is located in the GetNEList command response. The default is **1**. Specifying this parameter is optional. |
| gatewayNEIDResponseFieldPosition | Identifies the field where the NE ID is located in the GetNEList command response. The default is **1**. Specifying this parameter is optional. |
| mode | Identifies the mode of the Record and Playback feature. Valid values are **Normal** (no record or playback), **Record** (recording mode enabled), and **Playback** (playback mode enabled). Specifying this parameter is optional. |
| recordFileDir | Identifies where the cartridge writes and retrieves recorded TL1 data files. Specifying this parameter is optional. |

In addition, the TL1 Property Initializer processor creates the command dictionary and loads it with all the command documents defined in the cartridge. The TL1 cartridge defines the following commands in the command dictionary:

- ACT-USER-REQUEST
- CANC-USER-REQUEST

See *Network Integrity Developer's Guide* for more information about command dictionaries. See "About the Command Dictionary for the TL1 Cartridge" for more information about using the command dictionary for TL1.

## TL1 Property Customizer

This processor is part of the Discover Abstract TL1 action.

This processor populates the TL1Properties object produced by the TL1 Property Initializer processor with the parameter values configured in the Network Integrity UI. Table 2–1, " TL1 Properties Initialized by the TL1 Property Initializer Processor", lists all the parameters that are customizable in the Network Integrity UI. The host property is initialized from the scan address.

### TL1 Connection Manager

This processor is part of the Discover Abstract Base TL1 action.

This processor takes the TL1Properties object and command dictionary produced by the TL1 Property Initializer processor to establish a TL1 connection. It provides a TL1Connection object to any extending cartridges and succeeding processors.

To establish a TL1 connection, this processor performs the following operations:

1. Creates a connection to the TL1 agent using the values in the TL1 Properties object.
2. Logs in to the TL1 agent using the credentials in the TL1 Properties object.
3. If a TL1 gateway is configured, issues the TL1 gateway command to retrieve the list of NEs.
4. If the command is successful, returns the TL1Connection object and the list of NE IDs.
5. When the scan ends, logs out and disconnects from the TL1 agent.

This processor logs any errors and fails the scan if it cannot establish a TL1 connection with the TL1 agent.

## About Record and Playback

The TL1 cartridge can be configured to record all discovered objects.

You enable the Record and Playback feature at run time by setting a managed bean (MBean) configured on a property group on the Discover Abstract TL1 action.

When recording mode is enabled, the TL1Connection object writes the response data to a TL1 data file in *Local_Dir*/*IP_HostName*/*Filename*.**rec**, where:

- *Local_Dir* is a local directory that you can configure in the MBean at runtime. If you do not set a value in the MBean, *Local_Dir* is set to *Domain_Home*/**tl1Data**, where *Domain_Home* is the Network Integrity domain.
- *IP_HostName* is the host property on the TL1 Properties object.
- *Filename* is either the command code passed to the TL1 Connection Manager processor or the full command string.

For example: *Local_Dir*/**10/156/66/191/***Filename*.**rec**

When playback mode is enabled, TL1 Connection reads the TL1 data file (created in Record mode and stored on the local hard drive) and sends the data back to the discovery cartridge without polling any network devices. The resource adapter does not require a connection to the network device.

For more information about enabling or disabling the Record and Playback feature, see "About Using Record and Playback".

You can explicitly disable recording mode on a specific command, such as on commands that contain sensitive and unencrypted information (for example, plain-text user names and passwords) by calling the TL1Connection setRecordDisabled method.

The Record and Playback feature is not recommended for clustered environments because it relies on files being saved and loaded from the file system.

## About Address Validation

The Discover Abstract TL1 action expects a valid IP address, host name, or IP address range in the **Scope** field on the **Scope** tab of the Network Integrity UI.

Any Discovery action extending the Discover Abstract TL1 action should use an address handler to validate entered addresses, such as the IPAddressHandler cartridge provided with Oracle Communications Design Studio.

## About Dependent and Independent TL1 Commands

The TL1 cartridge can send independent TL1 commands and can send commands that are dependent on the results of another TL1 command.

For example, the TL1 cartridge can send an independent TL1 command to get the name and model type of a device. It can then send a dependent command to get additional information about the device depending on the device type:

- If the device type is 15454, the TL1 cartridge collects additional equipment details.
- If the model type is other than 15454, the TL1 cartridge does not collect additional equipment details.

You can send multiple independent and dependent commands. Response values are associated with their request.

## About the Command Dictionary for the TL1 Cartridge

The TL1 cartridge uses the command dictionary for formatting TL1 requests and parsing TL1 responses. The command dictionary is a framework for defining data dictionary structures for the command requests and responses. It also provides utilities for converting the populated request types into command strings and converting device responses into structured data holders.

The command dictionary uses structures defined in the Design Studio data dictionary to define command requests and responses. The Design Studio data dictionary generates an XML schema compiled into Java classes using XML beans. Use the Java request and response classes to issue TL1 requests and receive TL1 responses.

The data dictionary provides the following functionality:

- Defines the commands once so that a library of commands is available to be used in multiple cartridges.
- Validates that mandatory parameters are specified and that the request message is valid. The request definition can specify enumerated values, data types, and ranges to provide more validation.
- Provides a typed response for custom TL1 components so that parameters can be accessed based on name instead of position. For example, you can code **getAid()** rather than know that the AID is in parameter section 2, in field 3, and code **getSection(2).getField(3)**.

See "About the Cisco ONS 15454 TL1 Reference Cartridge" for example TL1 commands.

**Note:** Oracle recommends that you use the command dictionary when extending the Discover Abstract TL1 action. You can use the base WebNMS TL1 API methods instead of the command dictionary. You can send a complete TL1 command string with TL1Connection.send or you can use syncSend to send a WebNMS TL1Message. In both cases, a TL1Message is returned and you must extract lines, parameter blocks, and fields using WebNMS TL1 API operations. Refer to the WebNMS TL1 documentation for more information.

The TL1 cartridge defines structures in the data dictionary for TL1 message types: TL1Request and TL1Response. These must be used when defining new command dictionary Requests and responses for TL1.

## Sending Commands Using the Command Dictionary

To send commands using the command dictionary:

1. In Design Studio, define TL1Request in the data dictionary. The name of the structure should match the TL1 command name, and the command request name should end with **request**. See "TL1 Requests" for more information.
2. In Design Studio, define the matching TL1Response in the data dictionary. The name of the structure should match the TL1 command name, and the command response name should end with **response**. See "TL1 Response" for more information.
3. Run the **buildCommandSchema.xml** Ant script.

   **Note:** The script must be copied to the extension cartridge before you can run it. See "About the buildCommandSchema.xml Script" for more information.

4. Add the JAR file generated by the **buildCommandSchema.xml** script to the **/lib** directory in the project classpath.
5. Run the **buildCommandDocument.xml** Ant script.

   **Note:** The script must be copied to the extension cartridge before you can run it. See "About the buildCommandDocument.xml Script" for more information.

6. In the Collector Java implementation, add the generated command document into the run-time command dictionary by coding the following:

   ```
   request.getCommandDictionary().addCommandDictionaryDocument(new String[]
   {"command_doc.xml"});
   ```

   where *command_doc* is the name of the command document in the **src** directory.

7. In the Java code, build and populate the TL1 request object. For example:

   ```
   rtrvEqptRequest rtrvEqptRequest = RtrvEqptRequest.Factory.newInstance();
   rtrvEqptRequest.setCTAG("1");
   rtrvEqptRequest.setAID("myAID");
   ```

```
rtrvEqptRequest.setTID("myTID");
```

8. Send the command to the TL1 connection and assign the response.

```
RtrvEqptResponse eqptResponse =
(RtrvEqptResponse)request.getTl1Connection().send(rtrvEqptRequest);
if (!TL1Constants.COMPLETION_
COMPLD.equalsIgnoreCase(eqptResponse.getCompletionCode())) {
  throw new ProcessorException(createErrorMessage(eqptResponse,
rtrvEqptRequest,
  request.getCommandDictionary()));
}
```

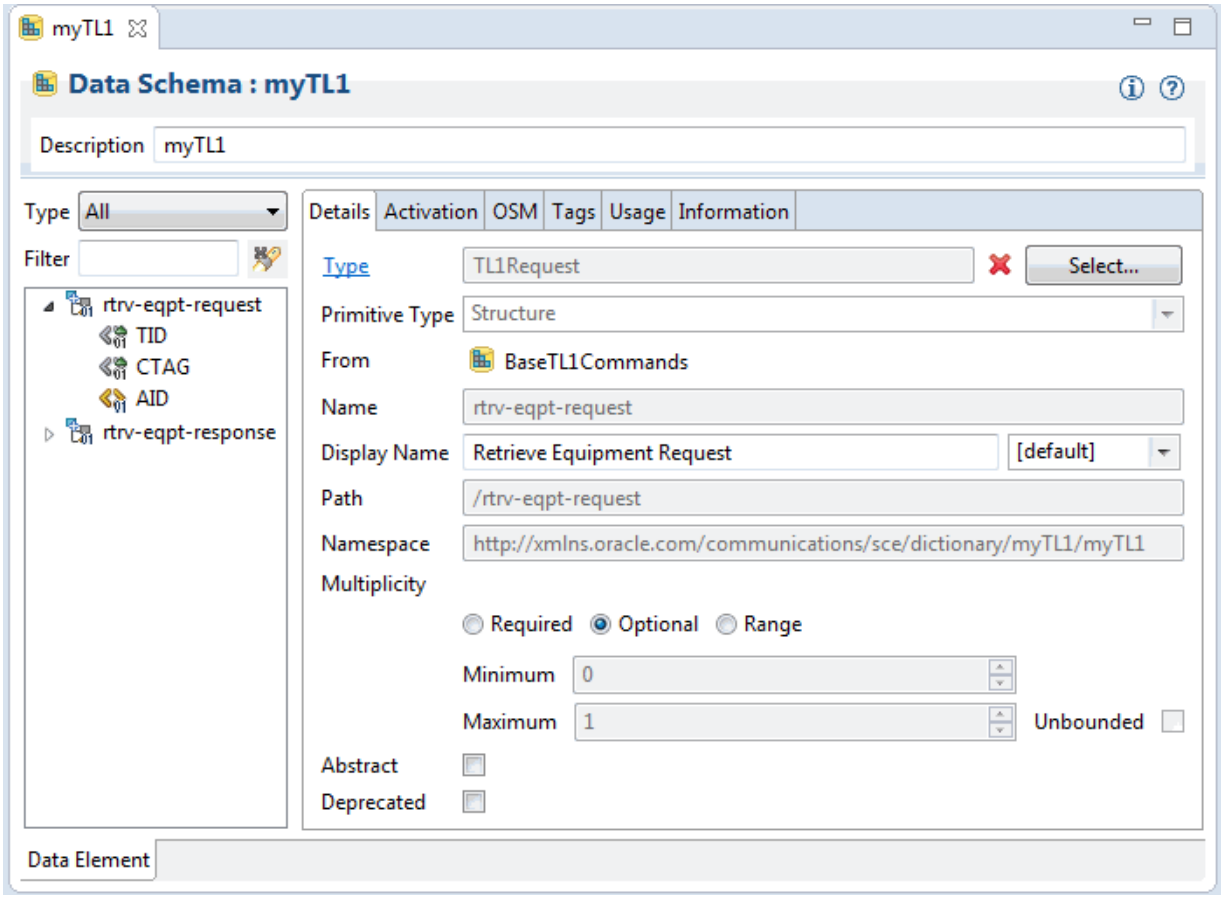
For more examples, refer to the Cisco 15454 TL1 reference cartridge, or see "Design Studio Extension".

## TL1 Requests

TL1 requests must use the TL1Request type as the base for the request structure. When creating the TL1 request, select **TL1Request** from the **Base** list. This type is defined in the TL1 Cartridge BaseTL1Commands data dictionary.

Figure 2–3 shows a request definition for retrieving equipment. The TID and CTAG attributes are inherited from the base TL1Request, but must add and define the AID attribute.

***Figure 2–3 Retrieve Equipment Request Definition***



For examples of TL1 requests, see"About the Cisco ONS 15454 TL1 Reference Cartridge".

## TL1 Response

TL1 responses must use the TL1Response type as the base for the response structure. When creating the TL1 response, select **TL1Response** from the **Base** list. This type is defined in the TL1 Cartridge BaseTL1Commands data dictionary.

> **Important:** The TL1 response must contain a single TL1ResponseLine child structure. Because TL1 commands return multiple lines of data, you must set the **Maximum** list of the TL1ResponseLine to **Unbounded**.

The TL1ResponseLine holds child parameter block structures that are of type TL1NamedParameterBlock or TL1PositionalParameterBlock. TL1 response parameter blocks are delimited by a colon. The TL1Line in the response must contain a child structure for each parameter block defined in the TL1 response.

For example, a typical TL1 Response is structured as follows (the number and type of parameter block defined is dependent on the TL1 command):

```
ResponseElement
  TL1Line (Maximum=Unbounded)
    TL1PositionalParameterBlock (Maximum=1)
      Parameter1
      Parameter2
      Parameter…n
    TL1PositionalParameterBlock (Maximum=1)
      Parameter1
      Parameter2
      Parameter…n
    TL1NamedParameterBlock (Maximum=1)
      Parameter1
      Parameter2
      Parameter…n
```

In the TL1 specification, parameter blocks have either positional fields or named fields. Positional fields place importance on the position of the value. Named fields consist of name-value pairs.

The following positional parameter block consists entirely of values, and does not contain any field names. Empty values are represented by consecutive commas.

```
SLOT-15,OC3-IR-4,,,1546.12
```

The following named parameter block contains name-value pairs.

```
PLUGTYPE=SX-IR-SW-SN,PN=87-31-00002,HWREV=004K,
FWREV=76-99-00009-004A,SN=013510,CLEI=NOCLEI,TWL1=1546.12,TWL2=1546.92,
TWL3=1547.72,TWL4=1548.51,PLUGINVENDORID=012345,PLUGINPN=ABCDE,
PLUGINHWREV=ABCDE,PLUGINFWREV=01-02-03,PLUGINSN=01234,ILOSSREF=1.0, PID=CISCO_
ONS15454,VID=V01,FPGA=F451,MODULETYPE=101
```

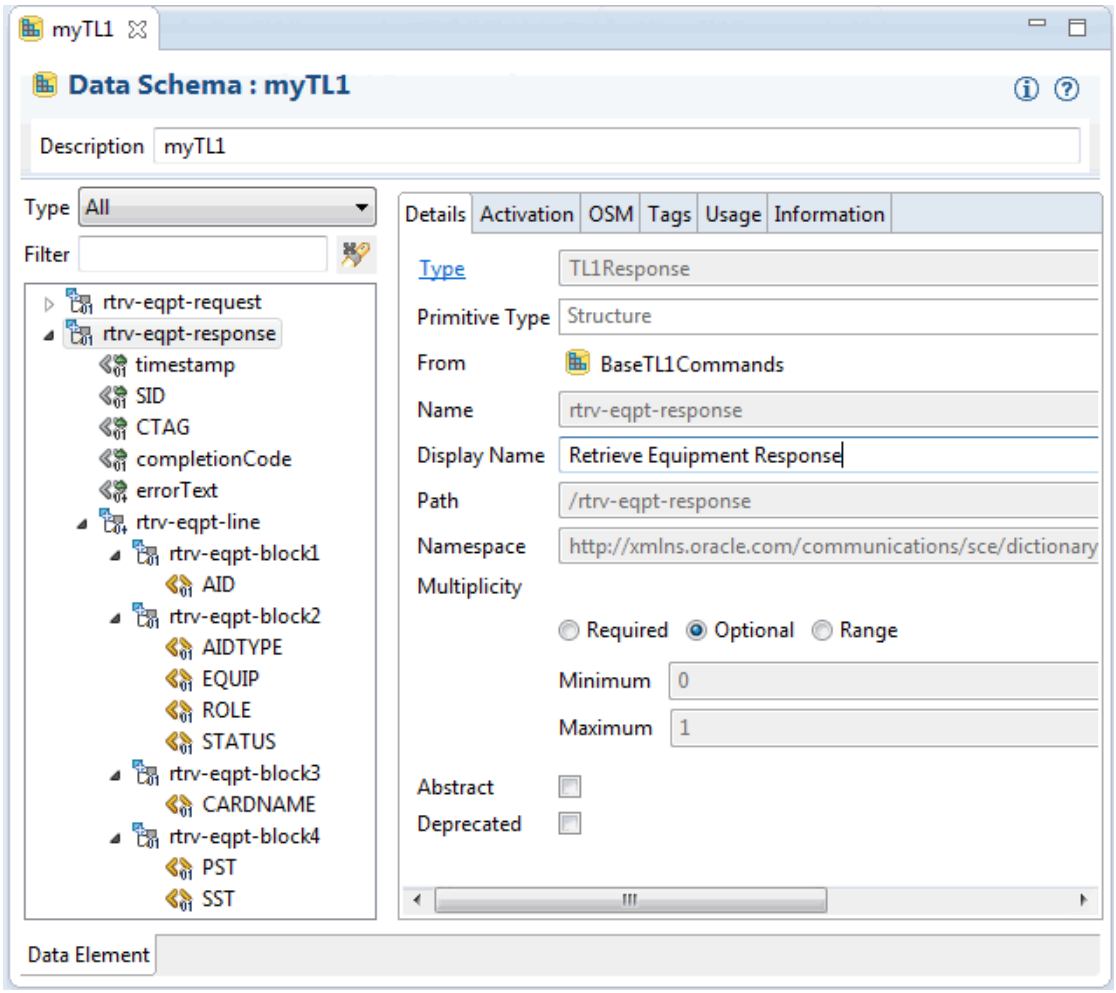
If a parameter block consists of positional fields and named fields, it is treated as a positional block.

For the TL1 response to be correctly populated, choose the correct parameter block when creating the TL1 response in the data dictionary.

Figure 2–4, "Retrieve Equipment Request Definition" shows a response definition for a Retrieve Equipment request, demonstrating the following elements:

- Structure rtrv-eqpt-response is of type TL1Response.
- The attributes timestamp, SID, CTAG, completionCode, and errorText are inherited from the base TL1Response.
- The timestamp is the date and time the response was returned.
- The SID is the NE Source Identifier returned by the TL1 agent.
- The CTAG is the Correlation tag value returned from the TL1 agent.
- The completion code contains the completion code returned by the TL1 agent. These are standard TL1 completion codes: COMPLD, DELAY, DENY, PRTL, RTRV.
- The errorText field contains any error messages returned in the TL1 response from the agent. If the request completed successfully, the errorText field is empty.
- The child structure responseLine is of type TL1Line and the Maximum attribute is set to **Unbounded** to handle multiple lines returned in the TL1 agent response.
- The child structures block1 and block2 are of type TL1PositionalParameterBlock because the AID, AIDTYPE, EQUIP, ROLE, and STATUS are not named fields in the response.
- The block3 group structure is of type TL1NamedParameterBlock because all fields are named. If the names of the fields do not match the name of the field returned by the device, the field will not be mapped. You should specify every field in the named parameter blocks, because the command response definition may be used by another cartridge.

***Figure 2–4 Retrieve Equipment Request Definition***




Some complex TL1 commands return multiple structured lines that contain differing command blocks. The default TL1 response mapper provided with the TL1 cartridge does not support this type of response; however, you can treat this scenario in one of the following ways:

- Provide a custom response mapper class that can handle the different line definitions. See *Network Integrity Developer's Guide* for more information.
- Use the generic WebNMS TL1 API instead of the command dictionary.
- Create a super-set of all the returned fields. If line 1 has fields A and B, and line 2 has fields C and D, define a single parameter block that has all four fields A, B, C, and D. On some lines, the A and B are populated; on other lines, C and D are populated. This solution only works with named parameter blocks.

The TL1 cartridge defines two commands in the command dictionary:

- ACT-USER: for logging in to the TL1 Session.
- CANC-USER: for logging out of the TL1 Session.

For examples of TL1 responses, see "About the Cisco ONS 15454 TL1 Reference Cartridge".

## About the buildCommandSchema.xml Script

The **buildCommandSchema.xml** Ant script compiles any schema (XSD) files ending in **Commands.xml** found in the **datadictionary** directory. The schemas are compiled using the XML Beans framework. The generated *Cartridge_Name***Commands.jar** file is written to the **/lib** directory. Add the *Cartridge_Name***Commands.jar** file to the project classpath before running the **buildCommandSchema.xml** script.

If the script does not find any schemas when it is run, the script does nothing.

The **buildCommandSchema.xml** script is located in the **base** directory of the TL1 cartridge project.

When extending the TL1 cartridge, copy the **buildCommandSchema.xml** script to the **base** directory of the extending cartridge.

Running the **buildCommandSchema.xml** script is a prerequisite to running the **buildCommandDocument.xml** Ant script.

## About the buildCommandDocument.xml Script

The **buildCommandDocument.xml** Ant script compiles command documents based on schema (XSD) files ending in **Commands.xml** found in the **datadictionary** directory. The generated *Data_Dictionary_Name***.xml** file (for example, **CiscoTL1Commands.xml**) is written to the **src** directory. The script looks for types in the schema that end with the word "request" or "response" and correlates them into a single command. See "About the Command Document and Command Document Templates" for more information.

The **buildCommandDocument.xml** script is located in the **base** directory of the TL1 cartridge project.

When extending the TL1 cartridge, copy the **buildCommandDocument.xml** script to the **base** directory of the extending cartridge.

The JAR file produced by the **buildCommandSchema.xml** script must be added to the project classpath before you can run the **buildCommandDocument.xml** script.

You can create a template file to override any of the values generated in the command document. The template contains the values to override. See "About the Command Document and Command Document Templates" for more information.

Add the generated command document to the command dictionary in the runtime operation, as shown in the following example:

```
request.getCommandDictionary().addCommandDictionaryDocument(new String[]
{"Cartridge_NameCommands.xml"});
```

Enter this code into the processor implementation before sending the TL1 commands.

If the script does not find any schemas ending in **Commands.xml**, the script does nothing.

## About the Command Document and Command Document Templates

You can create a command document template file to consistently override specific values generated in the command document. The **buildCommandDocument.xml** script automatically looks for such a template file.

The command document template file name must consist of the command file name and end in **-template.xml**. For example, if the command file name is

**CiscoTL1Commands.xml**, name the template file **CiscoTL1Commands-template.xml**. Keep the template in the same directory as the command document.

The command document describes the list of supported commands. One or more command documents are added to the command dictionary to configure the supported commands. In the TL1 Cartridge, the command document is generated by the **buildCommandDocument.xml** Ant script. If the generated command document is incorrect, any field can be overridden by using a template.

The command document template matches the command document schema and structure exactly and is used as base values by the **buildCommandDocument.xml** script when creating the command document XML. As long as the **commandDef** name matches, the values specified in the template override the generated values.

You can specify the following information for each command in the command document template:

- name: A unique name for the command. By default, the data dictionary structure name (minus the request or response suffix) is used for the name.
- description (optional): A description of what the command does.
- commandTemplate: Defines the template for the command syntax. By default, this is a velocity template string, but it could be anything depending on the CommandRequestHandler class. The **buildCommandDocument.xml** script inserts the data dictionary structure name (minus the request or response suffix) in this field and appends any child elements as arguments to the command. The commandTemplate field is the most likely to need to be overridden in a template file.
- commandRequestType: Identifies the request type. By default, this is the request schema type and must be unique in the command dictionary. For example:

  ```
  http://xmlns.oracle.com/communications/sce/dictionary/
  CommandParserProject/Cisco+TL1+Commands/GetAlarmsRequest
  ```

  This field is generated by **buildCommandDocument.xml** and populated by the data dictionary type ending in **request**.
- commandResponseType (optional): The class name that holds the response data. Not all commands have responses. The value is used by the registered CommandResponseHandler interface. For example:

  ```
  com.oracle.communications.sce.dictionary.CommandParserProject.Cisco_TL1_
  Commands.GetAlarmsResponse
  ```

  This field is generated by **buildCommandDocument.xml** and populated by the data dictionary type ending in **response**.
- errorResponseType (optional): The class name that holds the error response data. Not all commands have responses. This field is not used by the TL1 cartridge.
- requestFormatter (optional): Implements the CommandRequestHandler interface and is responsible for validating the request object and merging the data in the request object with the command template and returning a command string. If a CommandRequestHandler interface is not specified at the command or dictionary level, a default implementation is used that assumes the request object is an XML object and the command template is a velocity template. The TL1 cartridge uses the default implementation of the CommandRequestHandler interface.
- responseParser (optional): Implements the CommandResponseParser interface and is responsible for populating the response object based on the response string

provided. If a responseParser is not specified at the command level or the dictionary level, the response is not parsed. For the TL1 cartridge, there is a single CommandResponseHandler called oracle.communications.integrity.tl1cartridge.TL1ResponseParser that can format any TL1 response.

## About the Command Dictionary API

As a general rule, you should not need to use the command dictionary API.

The addCommandDictionaryDocument() operation is used to add the command document into the runtime command dictionary, as in the following example:

```
request.getCommandDictionary().addCommandDictionaryDocument(new String[]
{"Cartridge_NameCommands.xml"});
```

The CommandDictionary class provides a general interface for:

- Adding command definitions to and removing command definitions from the command dictionary.
- Retrieving information about dictionaries and loaded commands, such as getDictionaryNames() and getCommandNames().
- Setting default formatters, parsers, resolver, and types, such as setDefaultResponseType() and setDefaultRequestFormatter().

The CommandHandler class provides methods for performing the following operations on a single command:

- Formatting requests.
- Parsing responses.
- Checking for errors and returning messages.

## Sending a Command Using the Command Dictionary

TL1 commands can be sent to the TL1 agent over the TL1 connection, and TL1 responses can be parsed, after the following tasks are completed:

- Define TL1 requests and responses in the Design Studio data dictionary.
- Run the **buildCommandSchema.xml** script to generate XML Beans JARs.
- Run the **buildCommandDocument.xml** to generate the command document XML file.
- Add the command document to the runtime command dictionary.

To send a command:

1. Add the tl1Connection to the processor input parameters.
2. Create the request object and populate the parameters. For example:

```
RtrvEqptRequest rtrvEqptRequest = RtrvEqptRequest.Factory.newInstance();
rtrvEqptRequest.setCTAG("1");
rtrvEqptRequest.setAID("myAID");
rtrvEqptRequest.setTID("myTID");
```

3. Send the request instance and cast the response object. For example:

```
RtrvEqptResponse response =
(RtrvEqptResponse)request.getTl1Connection().send(rtrvEqptRequest);
```

4. Check the completion code. For example:

```
if (!TL1Constants.COMPLETION_
COMPLD.equalsIgnoreCase(eqptResponse.getCompletionCode())) {
  throw new ProcessorException(createErrorMessage(eqptResponse,
rtrvEqptRequest, request.getCommandDictionary()));
}
```

5. Extract any data necessary for modeling. For example:

```
for (RtrvEqptLine line : eqptResponse.getRtrvEqptLineArray()) {
  String aid = line.getRtrvEqptBlock1().getAID();
  String equip = line.getRtrvEqptBlock2().getEQUIP();
  String status = line.getRtrvEqptBlock2().getSTATUS();
}
```

# SSH Login Behavior

The following sequence describes the SSH login behavior:

- If the SSH Known Host File Location parameter is not set, the TL1 cartridge accepts all public keys from target devices.
- If the SSH Known Host File Location parameter is set:
  - And Network Integrity cannot write to the specified directory, the behavior is to accept all public keys from target devices.
  - And Network Integrity can write to the specified directory, all public keys are stored in the directory for each connection to a device, and all public keys are verified when the cartridge reconnects with a device to ensure that keys match. If the keys do not match, the public key is replaced.

# Using the TL1 Cartridge

This section describes how to use the TL1 cartridge after it is deployed to the server.

## Creating a Discovery Scan Action Type for TL1 Devices

You can create a discovery scan to discover TL1 device information in one or more network systems.

The TL1 cartridge has the following scan parameter groups:

- TL1Parameters: A group of the most commonly configured parameters for TL1 devices.
- TL1AdvancedParameters: A group of additional parameters that do not need to be configured for most TL1 devices because the default values are accepted by most.

To create a TL1 discovery scan, follow the instructions explained in the Network Integrity Help and do the following during the creation process:

1. On the **General** tab, do the following:
   a. From the **Scan Action** list, select the name of your TL1 Discovery action.

      The **Scan Type** field displays **Discovery**.

   b. Configure the following mandatory parameters:
      - In the **User Name** field, enter the user ID to start the TL1 session.

- In the **Password** field, enter the password for the user ID.
- In the **Port** field, enter the TL1 connection port.
- In the **Timeout** field, enter the timeout length of the underlying socket connection in seconds.
- In the **Retries** field, enter the number of retries that the cartridge attempts after a timeout.

**c.** (Optional) Configure the following optional parameters:

- To retry to connect after a timeout, enter the number of retries in the **Retries** field.
- To specify the transport protocol used by the TL1 device, select a protocol from the **Transport Protocol** list.
- If you selected **SSH** from the **Transport Protocol** list, enter the SSH session user ID and password in the **SSH User** and **SSH Password** fields.
- To specify where TL1 server SSH public keys are stored, enter a directory path in the **SSH Known Host File Location** field. All public keys are accepted when this field is left blank.

See Table 2–3, " TL1 Cartridge Scan Parameter Groups" for more information.

**d.** (Optional) Configure the advanced TL1 parameters:

- To identify the routing information for Network Integrity to send login messages to an NE, enter the target device in the **Login Target Identifier (TID)** field.
- To correlate an input command with an output response, enter the TL1 identifier in the **Login CTAG** field.
- To indicate whether the TL1 discovery scan is being used to discover the NEs from specific gateways, select **True** from the **Gateway Discovery** list.
- If the **Gateway Discovery** list is set to **True**, configure the **Gateway NE List Command** field, **Gateway NE Filter** field, **Gateway NEFilter Qualifier** list, **Gateway Response ID Section** field, and **Gateway Response ID** field to filter the TL1 gateway being discovered.

See Table 2–4, " TL1 Cartridge Advanced Scan Parameter Groups" for more information.

**2.** On the **Scope** tab, do one of the following:

- Enter an IP address.
- Enter a range of IP addresses (for example, **10.156.12.*** or **11.155.12.11-23** or **10.156.67.0/24**).
- Enter a host name.

The TL1 cartridge supports IPv4 and IPv6 IP address formats.

You can enter multiple addresses.

**3.** Make any other required configurations.

## About Using Record and Playback

The record feature allows you to record TL1 responses from devices for auditing, demonstration, or debugging the cartridge. The playback feature allows you to replay recorded files to simulate interaction with the device.

When record mode is enabled, the raw TL1 responses are written to a TL1 data file stored on the server.

When playback mode is enabled, the TL1 connection reads the TL1 data file (created in Record mode and stored on the server) and sends the data back to the discovery cartridge. The resource adapter does not require a connection to the network device.

Record and Playback is not recommended for clustered environments because it relies on files being saved and loaded from the file system.

A property group on the Discover Abstract TL1 action controls the Record and Playback feature. MBeans allow you to adjust the record and playback functionality in the runtime system without the need to restart systems or servers.

The Record and Playback feature mode property has three valid values:

- Normal: The Record and Playback feature recording mode and playback mode are disabled.
- Record: The Record and Playback feature recording mode is enabled.
- Playback: The Record and Playback feature playback mode is enabled.

### Viewing and Configuring the Current Record and Playback Mode

The MBean Browser in Oracle Enterprise Manager is used to view the mode property of the Record and Playback feature. See *Network Integrity System Administrator's Guide* for more information.

To view the current Record and Playback feature mode:

1. In the MBean Browser, navigate to the **oracle.communications.integrity.ActionProperty.ActionProperties** MBean.
2. Run the listPropertyGroups operation.

   This operation lists the configurable property groups. The Returned Value table displays the current mode.

To configure the Record and Playback feature mode:

1. In the MBean Browser, navigate to the **oracle.communications.integrity.ActionProperty.ActionProperties** MBean.
2. Select the *Action_Name***:TL1 Property Initializer:RecordPlayback** property group, where *Action_Name* is the name of the extending action.
3. Run the listProperties action, using the full property group name in the argument.
4. Copy the *Action_Name***:TL1 Property Initializer:RecordPlayback:mode** string from the Return Value table.
5. Open the setProperty operation and paste the string into the **Property** field.
6. In the Value field, set the value of the Record and Playback feature to either **Normal**, **Record**, or **Playback**.
7. Click the **Invoke** button.

To set the location where the recorded data is saved, open the **recordFileDir** action property and specify the desired directory. The directory must exist on the server and must be accessible by the Oracle WebLogic Server user.

# Design Studio Construction

This section provides information about the composition of the TL1 cartridge from the Design Studio perspective.

## Actions

The following tables outline the Design Studio construction of the TL1 cartridge actions and associated components:

- Table 2–2, " TL1 Cartridge Actions"
- Table 2–3, " TL1 Cartridge Scan Parameter Groups"
- Table 2–4, " TL1 Cartridge Advanced Scan Parameter Groups"
- Table 2–5, " TL1 Cartridge Processors"
- Table 2–6, " TL1 Cartridge Data Dictionary Structure"

***Table 2–2 TL1 Cartridge Actions***

| Action Name | Result Category | Address Handler | Scan Parameter Group | Processors |
|---|---|---|---|---|
| Discover Abstract Base TL1 | Device | IPAddressHandler | N/A | ▪ TL1 Property Initializer<br>▪ TL1 Connection Manager |
| Discover Abstract TL1 | Device | IPAddressHandler | See Table 2–3 and Table 2–4. | ▪ TL1 Property Customizer |

***Table 2–3 TL1 Cartridge Scan Parameter Groups***

| Characteristic Name | Parameter Type | Description | UI Label |
|---|---|---|---|
| tl1user | Text field | The user ID to start the TL1 Session. It can consist of up to 10 alphanumeric case-sensitive characters. | User Name |
| tl1Password | Text field | The password associated with the user name. It can consist of up to 10 alphanumeric case-sensitive characters. | Password |
| tl1Port | Text field | The port used for the TL1 connection. Valid range is 1 through 65535. The default is **6252** (TL1 SSH). | Port |
| tl1Timeout | Text field | The timeout of the underlying socket connection in seconds. The default is 5 seconds. | Timeout (Seconds) |
| tl1Retries | Text field | The number of retries that are attempted after a timeout. The default is 0. | Retries |
| transportProtocol | Drop down | Identifies the TL1 protocol to use: TCP, SSH, TELNET or OTHER. The default is SSH. | Transport Protocol |

***Table 2–3 (Cont.) TL1 Cartridge Scan Parameter Groups***

| Characteristic Name | Parameter Type | Description | UI Label |
|---|---|---|---|
| sshUser | Text field | The user ID used to establish the SSH session. This field is mandatory if transportProtocol is set to **SSH**. | SSH User |
| sshPassword | Text field | The password used to establish the SSH session. This field is mandatory if transportProtocol is set to **SSH**. | SSH Password |
| sshKnownHostFileDir | Text field | The location where the public keys of the TL1 server are stored if SSH is used. Leave blank to accept all public keys. If a directory is provided, the Network Integrity server must have permission to write to it, otherwise the TL1 cartridge defaults to accepting all public keys. | SSH Known Host File Location |

***Table 2–4 TL1 Cartridge Advanced Scan Parameter Groups***

| Characteristic Name | Parameter Type | Description | UI Label |
|---|---|---|---|
| tl1LoginTID (optional) | Text field | Identifies the routing information necessary for a login message sent by an OS to reach an NE. | Login Target Identifier (TID) |
| tl1LoginCTAG (optional) | Text field | Correlates the login input command with a response. Default value is 1. | Login CTAG |
| tl1GatewayFlag | Drop down | A Boolean value to indicate whether the discovery target is a gateway. The default is false. | Gateway Discovery |
| tl1GatewayNEListCommand (optional) | Text field | The command to retrieve the list of NEs from the gateway. This field is mandatory if tl1GatewayFlag is set to **True**. | Gateway NE List Command |
| tl1GatewayNEFilter (optional) | Text field | A filter on the NE ID returned from TL1 gateway. | Gateway NE Filter |
| tl1GatewayNEFilterQualifier (optional) | Drop down | This property works in combination with **Gateway NE Filter** to match the collected NEs by name and qualifier. | Gateway NE Filter Qualifier |
| gatewayNEIDResponseSectionPosition (optional) | Text field | Identifies where the NE ID is located in the GetNEList command response. The default value is **0**. | Gateway Response ID Section |
| gatewayNEIDResponseFieldPosition (optional) | Text field | Identifies the field where the NE ID is located in the GetNEList command response. The default value is **0**. | Gateway Response ID Field |

***Table 2–5 TL1 Cartridge Processors***

| Processor Name | Variable |
|---|---|
| TL1 Property Initializer | Input: N/A<br>Output:<br>■ tl1Properties<br>A properties class that holds the TL1 connection properties.<br>■ commandDictionary<br>The command dictionary containing the TL1 commands. |
| TL1 Property Customizer | Input:<br>■ tl1Properties<br>Customized with values from scan parameter groups.<br>Output: N/A |
| TL1 Connection Manager | Input: tl1Properties, commandDictionary<br>Output:<br>■ tl1Connection<br>An active connection to the TL1 agent specified in the TL1 Properties.<br>■ gatewayNEList<br>A list of NE IDs that were returned from the TL1 Gateway. Only the IDs that matched **Gateway NE Filter** are returned. If it is not a gateway discovery, there will be a single empty entry in the list, which is an empty string. This construction allows for the same set of processors inside a For Each processor to support gateway and non-gateway discovery.<br>■ gatewayNEDataMap<br>A map that contains the details of the NE response from the gateway. The key is the NE ID (that matches the value from gatewayNEList) and the value is the data retrieved from the gateway for the NE. If it is not a gateway discovery, the map will be empty. |

***Table 2–6 TL1 Cartridge Data Dictionary Structure***

| Element | Parameters | Notes |
|---|---|---|
| TL1Request | ■ TID (string)<br>■ CTAG (string) | The TL1Request is the base type of all TL1 Requests. It is expected that all TL1 command requests extend this type. |
| TL1Response | ■ timestamp (dateTime)<br>■ SID (string)<br>■ CTAG (string)<br>■ completionCode (string)<br>■ errorText (string) | The TL1Response is the base type of all TL1 Responses. It is expected that all TL1 command responses extend this type. |
| ACT-USER-REQUEST (TL1Request) | ■ TID (string)<br>■ CTAG (string)<br>■ userid (string)<br>■ password (string) | The TL1 request is used to log the user in to the TL1 session. |

***Table 2–6 (Cont.) TL1 Cartridge Data Dictionary Structure***

| Element | Parameters | Notes |
|---|---|---|
| ACT-USER-RESPONSE (TL1Response) | ▪ timestamp (dateTime)<br>▪ SID (string)<br>▪ CTAG (string)<br>▪ completionCode (string)<br>▪ errorText (string) | The TL1 response is the response login of the TL1 session. |
| CANC-USER-REQUEST (TL1Request) | ▪ TID (string)<br>▪ CTAG (string)<br>▪ userid (string) | The TL1 request is used to log the user off from the TL1 session. |
| CANC-USER-RESPONSE (TL1Response) | ▪ timestamp (dateTime)<br>▪ SID (string)<br>▪ CTAG (string)<br>▪ completionCode (string)<br>▪ errorText (string) | The TL1 response is the response from logging off the user from the TL1 session. |
| TL1ResponseLine | N/A | The base structure type for a TL1 Response Line. |
| TL1NamedParameterBlock | N/A | The base structure type for a named parameter block in a TL1 Response. All child elements must be named exactly as they are in the TL1 response. Not all child parameters from the TL1 message section must be specified. A subset is possible. |
| TL1PositionalParameterBlock | N/A | The base structure type for a positional parameter block in a TL1 Response. Position (order) of the child attributes is more important than the name of the attribute. Because position is important, all parameters should be specified, but parameters can be truncated off the end if they are not needed. |

# Design Studio Extension

This section contains examples and explanations about how to extend certain aspects of the TL1 cartridge. Refer to *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:

- Sending New Commands and Model Results
- Bypassing the Custom Banner for TL1 Devices

## Sending New Commands and Model Results

This example explains how to create a discovery action that sends new commands and model results defined in the command dictionary. In this example, the extending action models the collected data into the Information Model using the TL1Connection and TL1 commands defined in the command dictionary.

To create a discovery action that sends new commands and model results:

1. Open Design Studio in the Design perspective.
2. Create a new cartridge project called SampleTL1.

3. Open the Package Explorer view and expand **TL1_Cartridge**.
4. Copy **buildCommandDocument.xml** and **buildCommandSchema.xml** from the **TL1_Cartridge** directory and paste them in the **SampleTL1** directory.
5. In your SampleTL1 cartridge project, create a Discovery action called **Discover Sample TL1** and extend it with the Discover Abstract TL1 action.
6. Create a data dictionary called **SampleTL1Commands** to hold the TL1 command request and response definitions.

   Ending the dictionary name with **Commands** ensures that it automatically builds with the run script.
7. In **SampleTL1Commands**, define new TL1 request structures:
   - Each new TL1 request structure must extend the TL1Request structure: select **TL1Request** from the **Base** list.
   - Each TL1 request structure name should end with **Request** so that the run script can generate a correct command template for the requests and responses.

     For example, define a structure named **rtrv-eqpt-request**.
   - Add a child element called **AID**.

   See the Cisco 15454 TL1 reference cartridge for an example command request structure.
8. In **SampleTL1Commands**, define new TL1 response structures:
   - Each new TL1 response structure must extend the TL1Response structure: select **TL1Response** from the **Base** list.
   - Each TL1 response structure name should end with **response** so that the run script can generate a correct command template for the requests and responses.

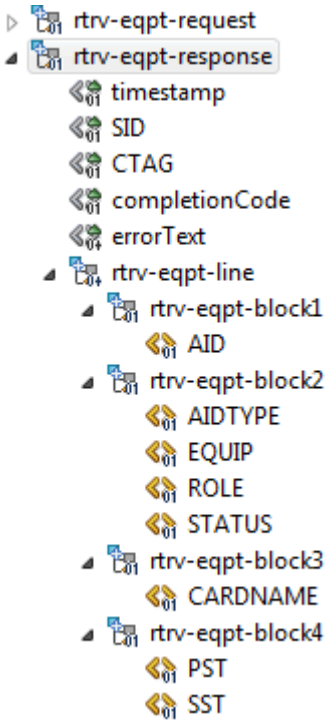
     For example, define a structure named **rtrv-eqpt-response**.

   See the Cisco 15454 TL1 reference cartridge for an example command response structure.

   **Note:** The response structure must match the response structure being returned by the TL1 agent.
9. Under **rtrv-eqpt-response**, add a child structure called **rtrv-eqpt-line**:
   - From the **Type** list, select **TL1ResponseLine**.
   - From the **Maximum** list, select **Unbounded**.
10. Under **rtrv-eqpt-line**, add a child structure called **rtrv-eqpt-block1**:
    - From the **Type** list, select **TL1PositionalParameterBlock**.

      The position of the response parameter is significant for this block. See "TL1 Response" for more information about child structure types and parameter blocks.
    - Add a child element called **AID** and set the **Type** list to **String**.
11. Continue to add parameter blocks until the response definition resembles Figure 2–5, then save your changes.

***Figure 2–5 Response Structure***




12. Open the Package Explorer view and run the **buildCommandSchema.xml** Ant script.

    The **buildCommandSchema.xml** file builds the data dictionary schema using XML Beans and creates **SampleTL1Commands.jar** in the **/lib** directory.

13. Add **SampleTL1Commands.jar** to the build path.

14. Run the **buildCommandDocument.xml** Ant file.

    The **buildCommandDocument.xml** file creates **SampleTL1Commands.xml** in the **/src** directory.

15. Open **SampleTL1Commands.xml** and make note of the commandTemplate field value.

16. Close **SampleTL1Commands.xml**.

17. Create a command document template by making a copy of SampleTL1Commands.xml and naming it **SampleTL1Commands-template.xml**.

18. Open **SampleTL1Commands-template.xml** and modify the CTAG value from **$!cmd.CTAG** to **12345**. Also, delete the requestType and responseType elements because those values are not being overridden.

19. Re-run **buildCommandDocument.xml**.

20. Open **SampleTL1Commands.xml** and verify that the commandTemplate field value is set to the version from the command document template.

21. Open the Discover Sample TL1 action editor.

22. On the **Processor** tab, add a For Each processor after the TL1 Connection Manager processor and select gatewayNEList as the parameter to iterate over. Name the iteration variable **neID**.

23. Create a new discovery processor named **Sample TL1 Collector** that creates and sends TL1 requests to collect data. Place it inside the For Each processor.
24. Open the **Sample TL1 Collector** processor.
25. On the **Context Parameters** tab and add the following parameters as input:
    - The commandDictionary parameter from the TL1 Property Initializer processor.
    - The tl1Connection parameter from the TL1 Connection Manager processor.
    - The neID parameter from the Discover Sample TL1 processor.
26. Add an output parameter:
    - In the **Parameter Name** field, enter **rtrvEqptResponse**.
    - In the **Parameter Type** field, enter **com.oracle.xmlns.communications.sce.dictionary.sampleTL1.sampleTL1Commands.RtrvEqptResponse**.
27. On the **Details** tab, click **Implementation Class**.

    The Java implementation for the processor is created.
28. Add the following code to the invoke method of the implementation class:

```
try {
    request.getCommandDictionary().addCommandDictionaryDocument(new String[]
{"SampleTL1Commands.xml"});
```

    This method adds the commands defined in the XML file to the runtime command dictionary.
29. Add the following code to the invoke method of the implementation class:

```
RtrvEqptRequest rtrvEqptRequest = RtrvEqptRequest.Factory.newInstance();
  rtrvEqptRequest.setCTAG("1");
  rtrvEqptRequest.setAID("myAID");
  rtrvEqptRequest.setTID("myTID");

  RtrvEqptResponse response =
(RtrvEqptResponse)request.getTl1Connection().send(rtrvEqptRequest);

  return new SampleTL1CollectorProcessorResponse(response);

} catch (Exception e) {
    throw new ProcessorException(e);
}
```

    This code creates a new RTRV-EQPT request, sets request arguments, sends the request to the TL1Connection, and gets a response. The RtrvEqptResponse instance is returned by the processor so that subsequent processors can use the value.
30. Open the Discover Sample TL1 action editor.
31. Create a new discovery processor called **Sample TL1 Modeler** to model the TL1 response data output from the Sample TL1 Collector processor. Place it in the For Each processor after the Sample TL1 Collector processor.
32. Open the **Sample TL1 Modeler** processor to the **Context Parameters** tab.
33. Add rtrvEqptResponse from the Sample TL1 Collector processor as an input parameter.

34. Design an implementation for the Sample TL1 Modeler processor that converts the TL1 responses to the Oracle Communications Information Model.
35. Build, deploy, and test your cartridge.

    The new processors are run in the order shown in Figure 2–6.

***Figure 2–6 Sending New Commands and Modeling Results Processor Workflow***


## Bypassing the Custom Banner for TL1 Devices

Certain TL1 devices display a banner after connecting, which is not handled by the default WebNMS telnet implementation. This example explains how to create a discovery action that extends the Discover TL1 action to bypass the custom banner.

For information about the WebNMS TL1 API specification, see the WebNMS documentation.

To bypass the custom banner for TL1 devices:

1. Create a new discovery action that extends the Discover TL1 action.
2. Create a new discovery processor called **TL1 Connection Customizer Initializer**. Place it before the TL1 Connection Manager processor.
3. Implement a WebNMS TL1 ConnectionHandler class, SkipBannerConnectionHandler. In the postConnect method, get the transport provider from the TL1Session and issue appropriate reads to bypass the banner.
4. In the new processor, register your customizer:

```
properties.setConnectionCustomizer(new Customizer() {
    @Override
    public void customize(TL1Connection connection) {
       connection.setConnectionHandler(new SkipBannerConnectionHandler());
    }
```

5. Build, deploy, and test your cartridge.

   The new processor is run in the order shown in Figure 2–7.

*Figure 2–7 TL1 Connection Customizer Extension Cartridge Workflow*

# 3

# About the Cisco ONS 15454 TL1 Reference Cartridge

This chapter describes the functionality and design of the Oracle Communications Network Integrity Cisco ONS 15454 SONET TL1 reference cartridge (Cisco 15454 TL1 reference cartridge) and how to use and build the cartridge.

## About Actions and Processors

The Cisco 15454 TL1 reference cartridge contains the Discover Cisco 15454 TL1 action. The Discover Cisco 15454 TL1 action discovers Cisco 15454 TL1 devices in your network. This discovery action extends the Discover Abstract TL1 action to establish the TL1 connection and set the TL1 properties.

The Discover Cisco 15454 TL1 action scans devices and provides a physical hierarchical model of the discovered data. This cartridge is designed to discover Cisco ONS 15454 SONET devices only. The scan fails if you use this cartridge to discover non-Cisco or other Cisco devices.

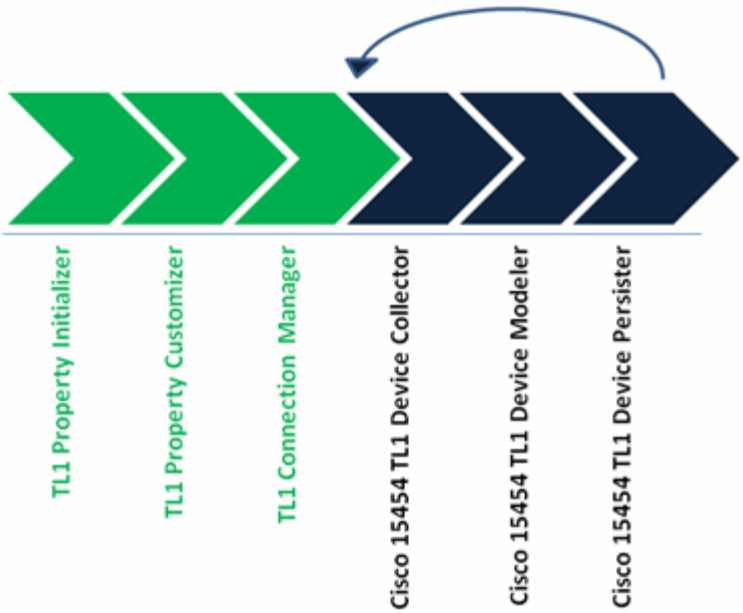
This discovery action inherits all the processors from the Discover Abstract TL1 actions. For more information about the inherited processors, see "About the TL1 Cartridge".

The Discover Cisco 15454 TL1 action contains the following processors run in the following order:

1. TL1 Property Initializer (inherited)
2. TL1 Property Customizer (inherited)
3. TL1 Connection Manager (inherited)
4. Cisco 15454 TL1 Device Collector
5. Cisco 15454 TL1 Device Modeler
6. Cisco 15454 TL1 Device Persister

Figure 3–1 illustrates the processor workflow of the Discover Cisco 15454 TL1 action.

*Figure 3–1 Discover Cisco 15454 TL1 Action Processors Workflow*




### Cisco 15454 TL1 Device Collector

This processor retrieves the device data using TL1 protocol and makes it available for modeling. This processor runs four TL1 commands for each device.

- RTRV-NETYPE: to get network element (NE) level details.
- RTRV-NE-GEN: to get NE level details.
- RTRV-INV: to get card level details.
- RTRV-EQPT: to get card level details.

### Cisco 15454 TL1 Device Modeler

This processor models the data collected by the Cisco 15454 TL1 Device Collector processor, building the hierarchical relationship for physical devices and children equipment, equipment holders, and physical ports.

### Cisco 15454 TL1 Device Persister

The Cisco 15454 TL1 Device Persister persists the physical device tree to the Network Integrity database.

## About Collected Data

This section discusses the data that is collected for modeling, listing, and explaining each request and response command.

### Equipment Collection

This section shows the TL1 request definition and sample TL1 responses that the processors could receive from a device.

The RTRV-NETYPE request resembles the following example:

```
RTRV-NETYPE:[<TID>]::<CTAG>;
```

The RTRV-NETYPE response line is made up of a positional parameter block with the following format:

```
SID DATE TIME
M CTAG COMPLD
"<VENDOR>,<MODEL>,<NETYPE>,<SW_ISSUE>"
;
```

Example 3–1 shows an example RTRV-NETYPE response.

***Example 3–1 RTRV-NETYPE Response Example***

```
TID-000 1998-06-20 14:30:00
M 001 COMPLD
"CISCO,ONS15454,ADM&MSPP&MSTP,5.00.00"
;
```

The RTRV-NE-GEN request resembles the following example:

```
RTRV-NE-GEN:[<TID>]::<CTAG>;
```

The RTRV-NE-GEN response line is made up of one named parameter block with the following format:

```
SID DATE TIME
M CTAG COMPLD
"[IPADDR=<IPADDR>],[IPMASK=<IPMASK>],[DEFRTR=<DEFRTR>],[IPV6ADDR=<ipv6addr>],[IPV6
PREFLEN=<ipv6preflen>],[IPV6DEFRTR=<ipv6defrtr>],[IPV6ENABLE=<ipv6enable>],[IIOPPO
RT=<IIOPPORT>],[NTP=<NTP>],[NAME=<NAME>],[SWVER=<SWVER>],[LOAD=<LOAD>],[PROTSWVER=
<PROTSWVER>],[PROTLOAD=<PROTLOAD>],[DEFDESC=<DEFDESC>],[PLATFORM=<PLATFORM>],[SECU
MODE=<SECUMODE>],[SUPPRESSIP=<SUPPRESSIP>],[MODE=<MODE>],[MSPUBVLANID=<MSPUBVLANID
>],[MSINTLVLANID=<MSINTLVLANID>],[AUTOPM=<AUTOPM>],[SERIALPORTECHO=<SERIALPORTECHO
>],[OSIROUTINGMODE=<OSIROUTINGMODE>],[OSIL1BUFSIZE=<OSIL1BUFSIZE>],[OSIL2BUFSIZE=<
OSIL2BUFSIZE>], [NET=<NET>] >],[BKUPNTP=<BKUPNTP>]"
;
```

Example 3–2 shows an example RTRV-NE-GEN response.

***Example 3–2 RTRV-NE-GEN Response Example***

```
TID-000 1998-06-20 14:30:00
M 001 COMPLD
IPADDR=192.168.100.52,IPMASK=255.255.255.0,DEFRTR=192.168.100.1,IPV6ADDR="[3ffe:05
01:0008:0000:0260:97ff:fe40:efab]",IPV6PREFLEN=64,IPV6DEFRTR="[3ffe:0501:0008:0000
:0260:97ff:fe40:e000]",IPV6ENABLE=NO,IIOPPORT=57970,NTP=192.168.100.52,NAME="NODEN
AME",SWVER=2.01.03,LOAD=02.13-E09A-08.15,PROTSWVER=2.01.02,PROTLOAD=02.12-E09A-09.
25,DEFDESC=\"NE DEFAULTS
FEATURE\",PLATFORM=15454-ANSI,SECUMODE=NORMAL,SUPPRESSIP=YES,PROXYSRV=N,FIREWALL=N
,MSPUBVLANID=1,MSINTLVLANID=2,AUTOPM=NO,SERIALPORTECHO=Y,OSIROUTINGMODE=ES,OSIL1BU
FSIZE=512,OSIL2BUFSIZE=512"
;
```

The RTRV-INV request resembles the following example:

```
RTRV-INV:[<TID>]:<AID>:<CTAG>[::::];
```

The RTRV-INV response line is made up of three parameter blocks: a positional block, followed by an empty block, followed by a named block. The response line has the following format:

```
SID DATE TIME
M CTAG COMPLD
"<AID>,<AIDTYPE>::[PLUGTYPE=<PLUGTYPE>],[PN=<PN>],[HWREV=<HWREV>],[FWREV=<FWREV>],
[SN=<SN>],[CLEI=<CLEI>],[TWL=<TWL>],[PLUGINVENDORID=<PLUGINVENDORID>],[PLUGINPN=<P
LUGINPN>],[PLUGINHWREV=<PLUGINWREV>],[PLUGINFWREV=<PLUGINFWREV>],[PLUGINSN=<PLUGIN
SN>],[ILOSSREF=<ILOSSREF>],[PID=<PID>],[VID=<VID>],[FPGA=<FPGA>],[MODULETYPE=<MODU
LETYPE>]"
;
```

Example 3–3 shows an example RTRV-INV response.

***Example 3–3 RTRV-INV Response Example***

```
TID-000 1998-06-20 14:30:00
M 001 COMPLD
"SLOT-15,OC3-IR-4::PLUGTYPE=SX-IR-SW-SN,PN=87-31-00002,HWREV=004K,FWREV=76-99-0000
9-004A,SN=013510,CLEI=NOCLEI,TWL1=1546.12,TWL2=1546.92,TWL3=1547.72,TWL4=1548.51,P
LUGINVENDORID=012345,PLUGINPN=ABCDE,PLUGINHWREV=ABCDE,PLUGINFWREV=01-02-03,PLUGINS
N=01234,ILOSSREF=1.0,PID=CISCO_ONS15454,VID=V01,FPGA=F451,MODULETYPE=101"
;
```

The RTRV-EQPT request resembles the following example:

```
RTRV-EQPT:[<TID>]:<AID>:<CTAG>[::::];
```

The RTRV-EQPT response line is made up of four parameter blocks:

- Block 1 uses the TL1PositionalParameterBlock format and models the following element:
  - AID
- Block 2 uses the TL1PositionalParameterBlock format and models the following elements:
  - AIDTYPE
  - EQUIP
  - ROLE
  - STATUS
- Block 3 uses the TL1NamesParameterBlock format. This block does not need to model any values. The CARDNAME element is modeled in this block as a container.
- Block 4 uses the TL1PositionalParameterBlock format and models the following element:
  - PST
  - SST

The response line has the following format:

```
SID DATE TIME
M CTAG COMPLD
"<AID>:<AIDTYPE>,<EQUIP>,[<ROLE>],[<STATUS>]:[<PROTID>],[<PRTYPE>],[<RVRTV>],[<RVT
M>],[<CARDNAME>],[<IOSCFG>],[<CARDMODE>],[<PEERID>],[<REGENNAME>],[<PEERNAME>],[<T
RANSMODE>],[<RETIME>],[<SHELFROLE>],[<FRPROLE>],[<FRPSTATE>],[<FRPHOLDOFFTIME>],[<
ADMINCVLAN>],[<ADMINSVLAN>],[<CFMSTATE>],[<CCTIMER>],[<SWITCHWITHCRCALARM>],[<CRCT
HR>],[<CRCPOLLINTRVL>],[<CRCSOAKCOUNT>]:<PST>,[<SST>]"
;
```

**Note:** This format, taken from Cisco documentation, does not show the third parameter block as named block, but is inferred from the example response provided.

Example 3–4 shows an example RTRV-EQPT response.

***Example 3–4 RTRV-EQPT Response Example***

```
TID-000 1998-06-20 14:30:00
M 001 COMPLD
"SLOT-1:10GE-XP,UNEQUIP,,NA:CARDMODE=10GEXP-L2ETH,FRPROLE=SLAVE,FRPSTATE=DISABLED,
FRPHOLDOFFTIME=DISABLED,ADMINCVLAN=0,ADMINSVLAN=0,CFMSTATE=N,CCTIMER=ONE-SEC,SWITC
HWITHCRCALARM=Y,CRCTHR=10E-2,CRCPOLLINTRVL=60,CRCSOAKCNT=10:OOS-AU,AINS&UEQ"
"SLOT-2:GE-XP,UNEQUIP,,NA:CARDMODE=GEXP-L2ETH,FRPROLE=SLAVE,FRPSTATE=DISABLED,FRPH
OLDOFFTIME=DISABLED,ADMINCVLAN=0,ADMINSVLAN=0,CFMSTATE=N,CCTIMER=ONE-SEC,SWITCHWIT
HCRCALARM=Y,CRCTHR=10E-2,CRCPOLLINTRVL=60,CRCSOAKCNT=10:OOS-AU,AINS&UEQ"
;
```

# About Cartridge Modeling

The Cisco 15454 TL1 reference cartridge models collected data according to the Information Model. Collected data is modeled into the following entities:

- PhysicalDevice
- Equipment
- PhysicalPort
- EquipmentHolder

Static field model entities are compliant with version 1.0.1 of the Information Model. Dynamic field model entities are application specific.

## Field Mapping

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

- Table 3–1, " Physical Device Field Mapping"
- Table 3–2, " Equipment (Shelf) Field Mapping"
- Table 3–3, " Equipment (Card) Field Mapping"
- Table 3–4, " Equipment Holder Field Mapping"
- Table 3–5, " Physical Port Field Mapping"

***Table 3–1 Physical Device Field Mapping***

| Information Model Attribute | Information Model Support | TL1 Data Source | Type |
|---|---|---|---|
| ID | Static | N/A | String |
| Name | Static | RTRV-NE-GEN: NAME | String |
| Description | Static | RTRV-NE-GEN: DEFDESC | String |
| Serial Number | Static | N/A | String |
| Native EMS Name | Static | RTRV-NE-GEN: NAME | String |
| Physical Location | Static | N/A | String |

*Table 3–1 (Cont.) Physical Device Field Mapping*

| Information Model Attribute | Information Model Support | TL1 Data Source | Type |
|---|---|---|---|
| Software Version | Dynamic | RTRV-NE-GEN: SWVER | String |
| Protect Software Version | Dynamic | RTRV-NE-GEN: PROTSWVER | String |
| Model Name | Dynamic | RTRV-NETYPE: MODEL | String<br>Values: ONS15454, ONS15600, Unknown |
| NE Type | Dynamic | RTRV-NETYPE: NETYPE | String<br>Values: ADM, DCS, MSPP, MSTP |
| Discovered Vendor Name | Dynamic | RTRV-NETYPE: VENDOR | String |

*Table 3–2 Equipment (Shelf) Field Mapping*

| Information Model Attribute | Information Model Support | TL1 Data Source | Type |
|---|---|---|---|
| ID | Static | N/A | String |
| Name | Static | N/A | String<br>Programatically generated. |
| Description | Static | N/A | String |
| Serial Number | Static | N/A | String |
| Native EMS Name | Static | N/A | String<br>Programatically generated. |
| Physical Location | Static | N/A | String |

**Note:** A shelf is not an entity that can be directly discovered. Therefore, shelves do not have any characteristics defined beyond the existing entity attributes.

*Table 3–3 Equipment (Card) Field Mapping*

| Information Model Attribute | Information Model Support | TL1 Data Source | Type |
|---|---|---|---|
| ID | Static | N/A | String |
| Name | Static | RTRV-INV: AIDTYPE | String |
| Description | Static | N/A | String |
| Serial Number | Static | RTRV-INV: SN | String |
| Native EMS Name | Static | RTRV-INV: AIDTYPE | String |
| Physical Location | Static | N/A | String |
| Discovered Part Number | Dynamic | RTRV-INV: PN | String |
| Firmware Revision | Dynamic | RTRV-INV: FWREV | String |
| Hardware Revision | Dynamic | RTRV-INV: HWREV | String |

**Note:** Equipment is modeled only if RTRV-EQPT:EQUIP for that equipment is set to **EQUIP**.

Although the TL1 <AID> field is not modeled, it is used to determine the slot the equipment is mounted in.

***Table 3–4 Equipment Holder Field Mapping***

| Information Model Attribute | Information Model Support | TL1 Data Source | Type |
|---|---|---|---|
| ID | Static | N/A | String |
| Name | Static | N/A | String<br>Programmatically generated. SLOT-# to match the RTRV-INV: AID values. |
| Description | Static | N/A | String |
| Serial Number | Static | N/A | String |
| Native EMS Name | Static | N/A | String<br>Programmatically generated. SLOT-# to match the RTRV-INV: AID values. |
| Physical Location | Static | N/A | String |

**Note:** Equipment holders cannot be discovered directly, so they are automatically created and populated. Because the cartridge is for a Cisco ONS 15454 SONET device, the cartridge will create the shelf with 17 slots ready to be populated with card equipment. For other models, a mapping file can be used to list how many slots each model has based on the RTRV-NETYPE: MODEL field.

Because equipment holders are not discoverable, they are created without a specification.

***Table 3–5 Physical Port Field Mapping***

| Information Model Attribute | Information Model Support | TL1 Data Source | Type |
|---|---|---|---|
| ID | Static | N/A | String |
| Name | Static | N/A | String |
| Description | Static | N/A | String |
| Serial Number | Static | N/A | String |
| Native EMS Name | Static | N/A | String |
| Physical Location | Static | N/A | String |
| Port Number | Static | N/A | String |
| Custom Port Name | Static | N/A | String |

**Note:** Physical ports cannot be discovered directly, so they are automatically created and populated using the value provided in the Cisco15454Cards property group that maps card types to the number of ports. No ports are created on the parent card if no entry exists in the property group.

# Model Correction

Model correction occurs when the TL1 information received through discovery does not conform to the Information Model or is incomplete and therefore cannot be saved to the Network Integrity database.

The Cisco 15454 TL1 reference cartridge applies model corrections in the following cases:

- Missing EquipmentHolder under Equipment: TL1 does not directly discover slots. EquipmentHolder entities are created automatically under the shelf of the Cisco 15454. Because this Cisco 15454 shelf is specific to a model, the cartridge creates the 17 slots automatically.
- Missing Equipment (Shelf) under PhysicalDevice: TL1 does not discover card and node level details. The existence of a shelf is not explicit and must be derived.
- Missing PhysicalPort under Equipment: TL1 does not directly discover ports. PhysicalPort entities are created automatically under the cards of the Cisco 15454 depending on configured property values, as shown in the example below:

```
PhysicalDevice (Cisco 15454 Physical Device)
  Equipment (Cisco 15454 Shelf)
    EquipmentHolder (Specificationless entity)
      Equipment (Cisco 15454 Card)
        PhysicalPort (Specificationless entity)
```

# Using the Cisco 15454 TL1 Reference Cartridge

This section describes how to use the Cisco 15454 TL1 reference cartridge.

## Setting Up a Scan

The Cisco 15454 TL1 reference cartridge does not introduce any new scan parameter groups. Because the Cisco 15454 TL1 reference cartridge extends the TL1 cartridge, the configurable scan parameter groups are those from the TL1 cartridge.

## TL1 Gateway Discovery

The Cisco ONS 15454 SONET device can act as a TL1 gateway, through which the Cisco 15454 TL1 reference cartridge can discover other devices connected to the gateway. When discovering a TL1 gateway, you can configure and filter the list of returned NEs in the Network Integrity UI.

To discover TL1 devices through a Cisco ONS 15454 SONET TL1 gateway, create a TL1 discovery scan in the same way that you create a discovery scan for TL1 devices. See "Using the TL1 Cartridge" for the procedure. When setting the values in the **General** tab in the Scan Action Parameters area, do the following to set the Advanced TL1 scan parameter groups:

- Set the **Gateway Discovery** list to **True**.

- In the **Gateway NE List Command** field, enter the following TL1 request: **RTRV-MAP-NETWORK:[<TID>]::<CTAG>;**.

  The RTRV-MAP-NETWORK request collects the list of devices that are in the span of control of the device that is acting as a TL1 gateway.
- In the **Gateway Response ID Section** field, enter **0**.
- In the **Gateway Response ID** field, enter **1**.

The response line has the following format:

```
SID DATE TIME
M CTAG COMPLD
"<IPADDR>,<NODENAME>,<PRODUCT>"
;
```

Example 3–5 shows an example RTRV-MAP-NETWORK response.

***Example 3–5 RTRV-MAP-NETWORK Response Example***

```
TID-000 1998-06-20 14:30:00
M 001 COMPLD
"172.20.222.225,TID-000,15454"
;
```

## Property Groups

The Cisco 15454 TL1 reference cartridge uses a property group called Cisco15454Cards as a mapping table for the modeling processor to create physical ports on the cards. You can modify, add, or remove properties in this property group using the Enterprise Manager MBean Browser.

In the MBean Browser, this property group is found at the following location:

**Discover Cisco 15454 TL1:Cisco 15454 TL1 Device Modeler:Cisco15454Cards**

See *Network Integrity System Administrator's Guide* for more information about property groups and the MBean Browser.

Table 3–6 shows the preconfigured values for the Cisco15454Cards property group.

***Table 3–6 Cisco15454Cards Property Group Values***

| Name | Example Value |
|---|---|
| OC3IR4 | 4 |
| 10GEXP | 4 |

The discovered card name is taken from the AIDTYPE field in the RTRV-INV TL1 response. When adding new entries, remove spaces and dashes from the card name. For example, if the AIDTYPE is **OC3-IR-4**, enter the name in the property group as **OC3IR4**.

If the AIDTYPE of a card cannot be found in the table, no physical port is created under the card.

## Discovered Results

Discovered results have a result group for each device.

Table 3–7 lists the specifications that are used to model entities discovered by the Cisco 15454 TL1 reference cartridge in the Information Model.

***Table 3–7 Cisco 15454 TL1 Reference Cartridge Discovery Specifications***

| Specification | Information Model Entity |
|---|---|
| Cisco 15454 Physical Device | PhysicalDevice |
| Cisco 15454 Shelf | Equipment |
| Cisco 15454 Card | Equipment |

The tree below shows the hierarchy of the modeled devices:

```
PhysicalDevice (Cisco 15454 Physical Device)
    Equipment (Cisco 15454 Shelf)
        EquipmentHolder (Specificationless entity)
            Equipment (Cisco 15454 Card)
                PhysicalPort (Specificationless entity)
```

Physical device and shelf entities are modeled explicitly as Cisco model-specific specifications.

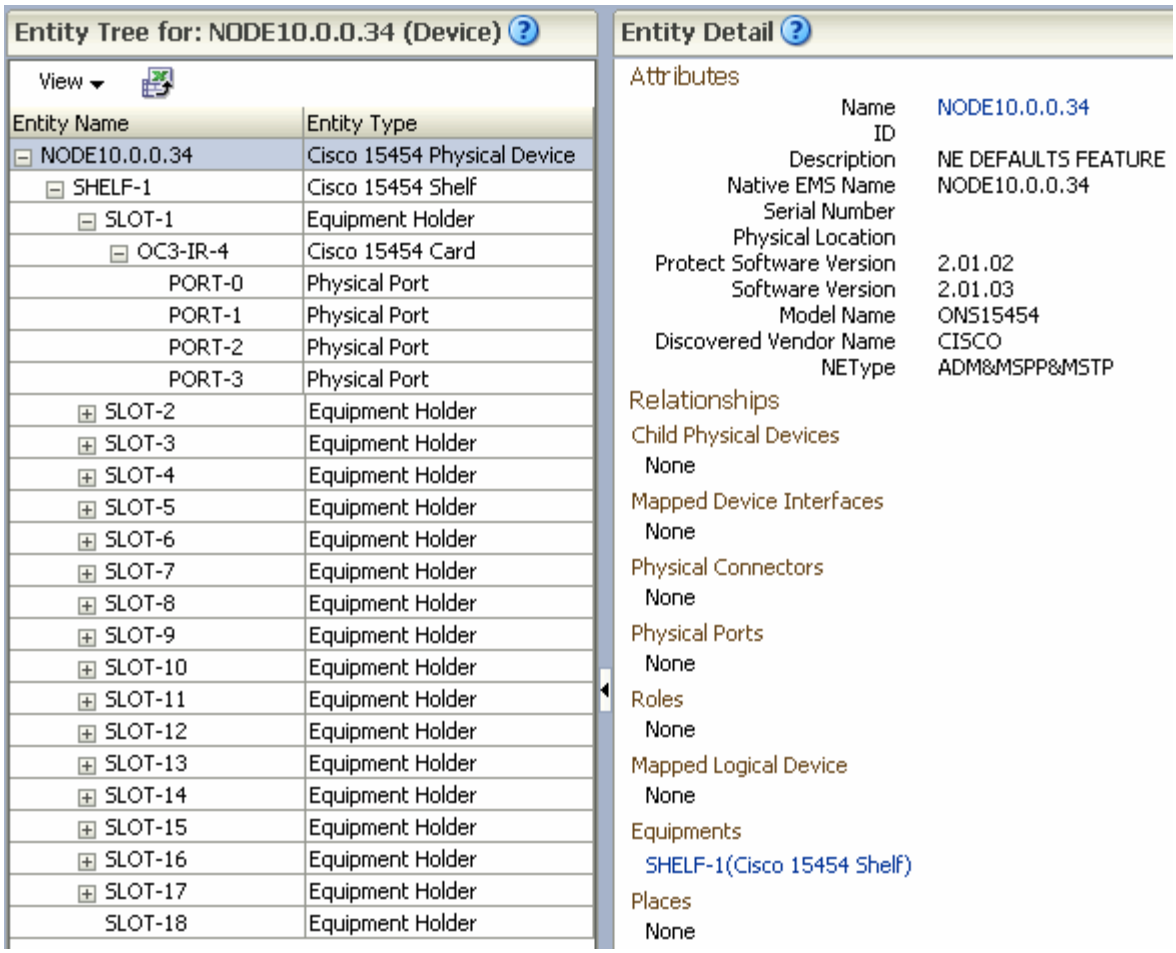
Cards are modeled using a generic Cisco 15454 card specification, instead of a card-specific specification. Depending on the inventory system used, you might want to explicitly model each card type as a specification.

Equipment holder (slot) entities are modeled without a specification because slots are not directly discoverable. Therefore, no parameters beyond the base entity attributes are populated. Because a slot is not a physical object, there is no need to explicitly define a specification for it. By creating a slot without a specification, you can extend the cartridge to assign any custom specification you require.

Ports are not directly discoverable. You can use and develop a property group to model ports on cards. See "Property Groups" for more information.

Figure 3–2 shows a sample set of discovered data from the Cisco 15454 TL1 reference cartridge.

*Figure 3–2 Sample Discovery Data From Cisco 15454 TL1 Reference Cartridge*




# Design Studio Construction

This section provides information about the composition of the TL1 cartridge from the Oracle Communications Design Studio perspective.

## Model Collections

Table 3–8 shows the Design Studio construction of the Cisco 15454 model collection.

*Table 3–8 Cisco 15454 Model Collection*

| Specification Name | Information Model Entity Type |
|---|---|
| Cisco 15454 Physical Device | PhysicalDevice |
| Cisco 15454 Shelf | Equipment |
| Cisco 15454 Card | Equipment |

## Actions

The following tables outline the Design Studio construction of the Cisco 15454 TL1 reference cartridge actions and associated components:

- Table 3–9, " TL1 Cartridge Actions"

- Table 3–10, " TL1 Cartridge Processors"
- Table 3–11, " Property Groups"

***Table 3–9 TL1 Cartridge Actions***

| Action Name | Result Category | Address Handler | Scan Parameter Groups | Processors |
|---|---|---|---|---|
| Discover Cisco 15454 TL1 | Device | IPAddress Handler | TL1 cartridge parameters. See Table 2–3 and Table 2–4. | ▪ TL1 Property Initializer<br>▪ TL1 Property Customizer<br>▪ TL1 Connection Manager<br>▪ Cisco 15454 TL1 Device Collector<br>▪ Cisco 15454 TL1 Device Modeler<br>▪ Cisco 15454 TL1 Device Persister |

***Table 3–10 TL1 Cartridge Processors***

| Processor Name | Variable |
|---|---|
| TL1 Property Initializer | See "About the TL1 Cartridge" for more information. |
| TL1 Connection Manager | See "About the TL1 Cartridge" for more information. |
| TL1 Property Customizer | See "About the TL1 Cartridge" for more information. |
| Cisco 15454 TL1 Device Collector | Input:<br>▪ neName<br>The name of the NE returned from the gateway command.<br>▪ tl1Connection<br>▪ gatewayNEDataMap<br>▪ tl1Properties<br>▪ commandDictionary<br>Output:<br>▪ eqptResponse<br>The response from RTRV-EQPT.<br>▪ invResponse<br>The response from RTRV-INV.<br>▪ netypeResponse<br>The response from RTRV-NEYPE.<br>▪ negenResponse<br>The response from RTRV-NE-GEN. |
| Cisco 15454 TL1 Device Modeler | Input: neName, gatewayNEDataMap, eqtResponse, invResponse, netypeResponse, negenResponse<br>Output: physicalTree |
| Cisco 15454 TL1 Device Persister | Input: physicalTree<br>Output: N/A |

*Table 3–11 Property Groups*

| Property Group | Type | Property | Processor |
|---|---|---|---|
| Cisco15454Cards | Managed and Map | OC3IR4 | Cisco 15454 TL1 Device Modeler |
| Cisco15454Cards | Managed and Map | 10GEXP | Cisco 15454 TL1 Device Modeler |

## Design Studio Extension

The source code to this cartridge is provided. You can change any part of the code to customize this cartridge to fit your environment, or you can use the code as an example on which to model your own custom TL1 device cartridge.

For more information about extensibility, see *Network Integrity Developer's Guide* and *Network Integrity Concepts*.