iReasoning TL1 Library User Guide

Table of Contents

Chapter 1

INTRODUCTION

About this document

The purpose of this document is to provide the reader with enough knowledge to start software development with the iReasoning's TL1 toolkit. Some examples are provided to better illustrate the usage of APIs. This document is not intended to provide detailed information about APIs. To effectively use APIs, readers need to refer to the javadoc help that has detailed instructions and examples.

The iReasoning TL1 API is a set of high performance Java library that greatly simply the development of applications for managing TL1 agents. The design and implementation of this library were based on object-oriented methodology and followed recognized design patterns. These advantages, combined with a highly optimized code base, make iReasoning TL1 library stand out from the competition.

Target Audience

This document is intended for users and engineers who are responsible for developing TL1 based management applications. User does not have to be an expert in network management field. However, She or he should be familiar with basic TL1 concepts. And some basic knowledge about java language is required to understand examples. One of the design goals of this library is to minimize the learning curve for users, make user write less code to achieve their programming job. So user will find it quite easy to learn and master this library.

Chapter 2

INSTALLATION

Requirements

- JDK1.2 or a later version must be installed. You can download JDK from the Oracle's web site (http://java.com)
- At least 32MB memory is required

Installation Procedures

1. Download and unzip

   Download iReasoning TL1 library and unzip it to desired directory, for example, C:\lib\iReasoning\TL1

   The directory structure will look like the following:

Directory Name	*Description*
lib	contains binary jar files
javadoc	contains javadoc HTML files
examples	contains examples source code
config	configuration files

2. Set up CLASSPATH

   ireasoningtl1.jar must be added to CLASSPATH environment variable. If SSH is required, ssh.jar also needs to added to the classpath.

   On Windows:

   If iReasoning TL1 library is installed at C:\lib\iReasoning\TL1, use the command

```
set CLASSPATH=C:\lib\iReasoning\TL1\lib\ireasoningtl1.jar;%CLASSPATH%
```

On Unix/Linux:

If iReasoning TL1 is installed at /usr/local/ireasoning/TL1, for C Shell, , use the command

```
CLASSPATH=/usr/local/ireasoning/TL1/lib/ireasoningtl1.jar ; export CLASSPATH
```

3. Run example program (Optional)

   To verify that everything is working, you can run the example programs.

   1. Add examples.jar to CLASSPATH. Take similar step, such as

      On Windows:

      ```
      set CLASSPATH=C:\lib\iReasoning\TL1\lib\examples.jar;%CLASSPATH%
      ```

      (assuming examples.jar is located at C:\lib\iReasoning\TL1\lib\ )

      On Unix/Linux

      ```
      CLASSPATH /usr/local/ireasoning/TL1/lib/examples.jar ;  export CLASSPATH
      ```

      (assuming examples.jar is located at /usr/local/ireasoning/TL1/lib/)

   2. Start TL1 Agent Simulator

      On Windows:

      Run runsimulator.bat

      On Unix/Linux

      ./runsimulator.sh

      A TL1 agent will be listening on port 9000.

      You can use

      ```
          telnet hostname 9000
      ```

      to verify that the agent is running.

   3. Open another shell prompt, run

      java tl1

Chapter 3

USING iREASONING TL1 LIBRARY

Overview and Architecture

iReasoning TL1 library is designed and implemented using object oriented methodology and recognized design patterns. One of the design goals is to minimize the learning curve of TL1 library for developers. Users only need to know a few classes, most of complex works are hidden from them. The example code shipped with this product clearly demonstrates the simplicity of this library. In those code, at first glance, user will realize how simple it is to write code to talk TL1 agent.

The central class is the TL1Session. A session represents a communication channel between TL1 manager and an agent. To communicate with multiple agents, multiple sessions have to be created. Each session is corresponding to each communication channel. Session needs to be created and then a connection needs to be established with the agent. After session is initialized, you are ready to issue TL1 commands against agent.

Both synchronous and asynchronous requests are supported in TL1Session. When sending synchronous requests, the session can only send one request at a time. It must block and wait for the reply to each request before issuing next one. This mode is easier to understand and implement. When using asynchronous requests, the session does not need to wait for each reply. In this case, session has another thread handle the reply and notify caller when reply arrives.

The remote agent is represented as Target class. Each agent is represented by a single Target object. An Target can be instantiated with the IP address (or host name) and port number of remote agent. The class diagram is shown in figure 1. For detailed information about classes, you can refer to the javadoc HTML files shipped with this library.

The messages from TL1 agent are represented by three classes: TL1ResponseMsg, TL1NotificationMsg and TL1AckMsg, all of them are derived from TL1OutputMsg. TL1NotificationMsg is agent-initiated autonomous message to report problems or significant events. TL1ResponseMsg is the reply message to the

TL1 command sent from manager. TL1AckMsg is sent when agent can not send the response message within a certain amount of time. Each TL1 message is fully parsed and translated to Java objects. For example, the date and time information contained in message header are represented by TL1Date and TL1Time objects. The payload part is represented as an array of TL1Line object. And from those objects, you can further drill down to get more detailed information.

UML diagram

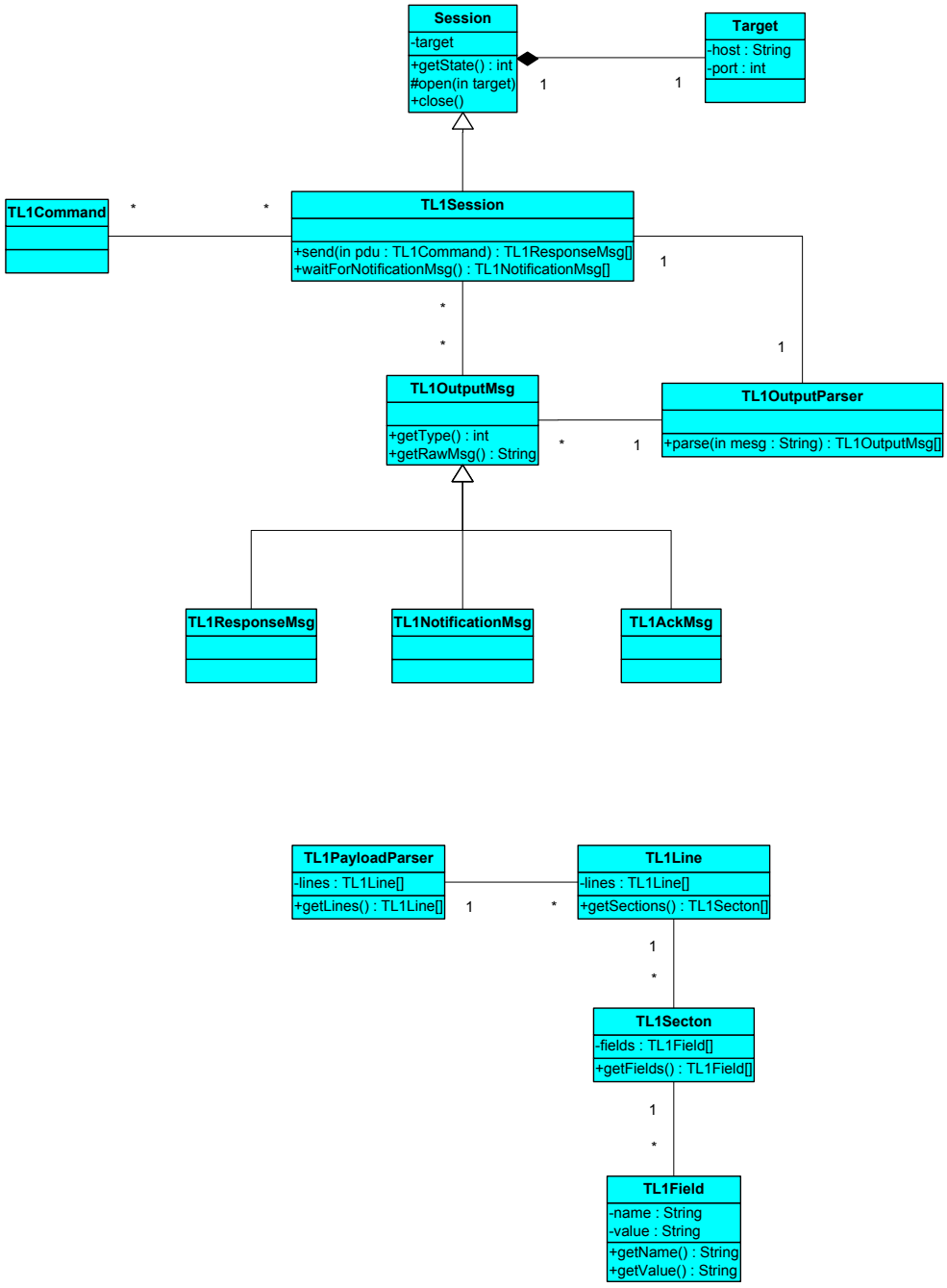

Figure 1. iReasoning TL1 library class diagram

Configuration

➢ Logger configuration

iReasoning TL1 library has a built-in logger. Logger is configurable through Logger.prop file located at config directory. You can change the logging level or just disable logger. The output of logging message is also configurable. It can be standard out or file. Check out Logger.prop for detailed information.

Usage examples

In your source file, you need to import the following iReasoning TL1 packages:

```
import com.ireasoning.protocol.*;
import com.ireasoning.protocol.tl1.*;
```

➢ TL1Session class

The core class of iReasoning TL1 library is TL1Session class. It represents a communication channel between manager and agent. Similar to other communication classes, it needs remote host name and port number so it can connect to remote agent, and the channel needs to be closed after it is finished. This class provides necessary methods to do TL1 queries. The lowest level method is "public TL1ResponseMsg[] send(TL1Command request)", which encodes and sends out TL1Command object to the agent.

The following is a simple example on how to use TL1Session class to log in to TL1 agent and then send a RTRV-EQPT command.

```
1.  TL1Session session = new TL1Session("localhost", 9000);
2.  TL1Command req = new TL1Command();
3.  req = TL1Command.act_user("abc", "123");
4.   session.send(req);
5.   req.setCommand("rtrv-eqpt:NodeA:SLOT-ALL:123;");
6.   TL1ResponseMsg[] msgs = session.send(req);
     ... //process messages
7.   session.close();
```

In this example, a TL1Session object is created with the properties of remote agent. Then it sends a login command, and a RTRV-EQPT command is sent after that. Check out tl1.java (in ./examples/tl1/) for complete implementation example.

➢ Create a TL1 alarm listener

TL1 alarms are initiated by agent to report problems or significant events. To collect TL1 alarms, a connection needs to be established with the agent. And log in with necessary security level. Then we can collect all alarms sent by agent. Some agents terminate idle connections. To avoid that, TL1Session's lanuchKeepAliveThread method launches a new thread which sends a dummy command to the agent periodically.

The following is a simple example on how to collect TL1 alarms.

```
1. TL1Session session = new TL1Session("localhost", 9000);
2. L1Command  req = TL1Command.act_user("abc", "123");
3. session.send(req);
4. session.launchKeepAliveThread("rtrv-eqpt:Node:SLOT-ALL:123;", 60);
5. while(true)
6. {
7.    TL1NotificationMsg[] notes = session.waitForNotificationMsg();
8.    for (int i = 0; i < notes.length ; i++)
9.    {
10.       System.out.println( "Got a new alarm:\r\n" + notes[i]);
11.   }
12.}
```

The first line creates a TL1Session connecting to a local agent at port number 9000. Line 2 and 3 send login command with user name and password. Line 4 launches a new thread which sends keep-alive command to agent periodically. Line 5 to 12 is an infinite loop waiting for TL1 autonomous messages. Check out alarmd.java for a complete implementation.

APPENDIX

TL1 agent simulator

A TL1 agent simulator is included in this library. It is a multi-thread program which can handle multiple clients simultaneously. Run runsimulator.bat (on windows) or runsimulator.sh (on unix) to start simulator. The default port number is 9000.

You can config the simulator by modifying couples of configuration files on config directory.

- Simulator.xml

  Here is an example:

  ```
  <TL1Simulator
  port="9000"
  configFile="TL1Simulator.xml"
  dataFile="TL1Data.txt"
  alarmFile="TL1Alarm.txt"
  alarmPeriod="10"/>
  ```

  It specifies that TL1 agent listens on port 9000. Its own config file is TL1Simulator.xml. It loads TL1 commands and responses from TL1Data.txt. And it loads TL1 alarms from TL1Alarm.txt. It sends alarms every 10 seconds.

- TL1Simulator.xml

  Here is an example:

  ```
   <user    name="abc"    password="123"  />
  ```

  It adds TL1 user whose name is abc and password is 123.

- TL1Data.txt

  It contains the raw string input and output TL1 agent.

  Here is an example:

  ```
  ; RTRV-CRS-ALL:NODE_1_9:::::;
  > IP 0
  <
    NODE_1_9 01-05-14 20:18:34
  M  0 COMPLD
  ;
  ```

You can add new commands and responses to this file.

- TL1Alarm.txt

  It contains the raw alarm message. TL1 simulator loads alarm data from this file, and it picks an alaram randomly and sends it out periodically.

  Here is an example:

```
   NODE2 2000-05-23 15:41:43
*C 329 REPT ALM OC12
   "FAC-14-1:CR,LOF,SA,,,,:\"Loss of Frame\",OC12"
;
```

  You can add new alarms to this file.

RESOURCES AND REFERENCE

- iReasoning Home Page

  http://www.iReasoning.com/

- TL1 FAQ

  http://www.tl1.com/library/TL1/Overview/FAQ.html

- TL1 misc. information

  http://www.tl1.com/

- JAVA web site

  http://java.com

GLOSSARY OF TERMS

JAVA

Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model.

TL1

Transaction Language 1 (TL1) is an ASCII or man-machine management protocol defined by Bellcore. It is used to manage most of the optical, broadband, and access equipment in North America.

TCP

TCP (Transmission Control Protocol) is a set of rules (protocol) used along with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. While IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet. TCP is known as a connection-oriented protocol, which means that a connection is established and maintained until such time as the message or messages to be exchanged by the application programs at each end have been exchanged.

Transport Layer

In the Open Systems Interconnection (OSI) communications model, the Transport layer ensures the reliable arrival of messages and provides error checking mechanisms and data flow controls. The Transport layer provides services for both "connection-mode" transmissions and for "connectionless-mode" transmissions. For connection-mode transmissions, a transmission may be sent or arrive in the form of packets that need to be reconstructed into a complete message at the other end.

UDP

UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP. Like the Transmission Control

Protocol, UDP uses the Internet Protocol to actually get a data unit (called a datagram) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end.

UML

UML (Unified Modeling Language) is a standard notation for the modeling of real-world objects as a first step in developing an object-oriented design methodology.