Chapter 10

JSystem Automation Platform Application Libraries

➢ **In this chapter…**

10.1 JSystem Automation Platform Application Libraries Overview

The following section provides detailed functional information about the Ignis custom drivers. The drivers are divided into two groups, core drivers and network drivers.

10.2 Application Drivers

The Application drivers are divided into four categories according to their functionality, they are:

1. **Core Drivers**
2. **Network Drivers**
3. **Web Drivers**
4. **Swing Drivers**

10.3 Core Drivers

The core drivers refer to the connecting layer of the driver (system objects) that enables the system to connect and communicate with the SUT. The illustration shows the relevant core drivers in their relative position to the layers of the JSystem Automation Platform.

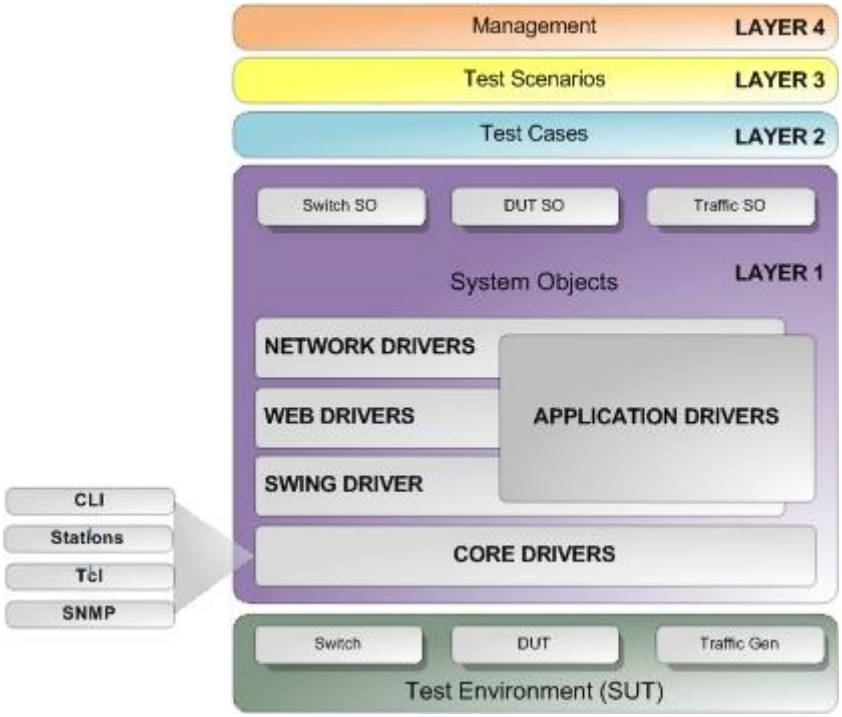

Figure 1: Core Drivers Layer

10.3.1 Ignis Cli Driver

10.3.1.1 Cli Overview

The command line interface (Cli) is a mechanism for interacting with a computer operating system or software application by typing commands to control the system, in contrasts with the use of a mouse pointer on a graphical user interface (GUI). This way of instructing a computer to perform a given task is referred to as "entering" a command: the system waits for the user to conclude the submitting of the text command by pressing the "Enter" key (a descendant of the "carriage return" key of a typewriter keyboard). A command line interpreter then receives, analyses, and launches the requested command.

The command line interpreter may be a text terminal or a remote shell client such as PuTTY.

Upon completion, the command usually returns output to the user in the form of text lines on the Cli. This output may be an answer if the command was a question, or otherwise a summary of the operation.

10.3.1.2 Ignis Cli Driver Overview

The **Ignis Cli Driver** provides a Java API that abstracts the work Cli protocols. The package has built in out of the box support for Telnet, SSH and RS232 and can be easily extended to support other Cli protocols. The API works with all operating systems that implement these protocols in a standard way.

The three main classes that the programmer works with are:

- **CliConnection** - Implements Cli client functionality: initiating the connection, processing commands (as described by **"CliCommand"** object), receiving command results and checking for command end by smart wait for "**Prompts**".
- **CliCommand** - Cli command descriptor, in addition to the command itself the descriptor can define command success and failure conditions, command time out, command re-send conditions and more (see "**CliCommand**" class javadoc).
- **CliCommand Prompt** - Descriptor for Terminal application prompt.

10.3.1.3 Cli Features

The following table details the list of features supported by the Cli module.

Function	Description
Out of the Box Protocols	SSh, Telnet , RS232
Architecture	Extensible Cli framework with out of the box support for windows XP/Vista and linux.
	Extensible infra structure for easy analysis of commands results that makes it easy to implement interactive session and to analyze success/failure of commands Out of the box analyzers for commonly performed analysis.
Session	Supports addition of traffic filters.
	Asynchronous messages handling.
	Configurable operation time out.
	Dump session to log file.
Prompts	Supports simple definition of prompts or complex regular expression based prompts.
Command	Easy definition of command success/failure conditions.
	Natural work with JSystem analyzers which enables easy addition of sophisticated results analyzers.

Table 1: Cli Features

10.3.1.1 Cli Specification Table

The following specification table details the WireShark driver operating system specifications.

Specification	Description	
Supported Versions	0.99.4	
Operating Systems	All operating systems that support standard Telnet/SSH/RS232	Out of the box support for Vista, Windows XP, Linux
Pre Requisite	Cli agent (Telnet/SSh/RS232) has to be active on the controlled machine	

Table 2: Cli Driver Specifications

10.3.1.2 Cli Code Example

The Cli code example shows that the code of a system object which implements several common Cli operations uses the Cli API.

```
01 package com.aqua.services.demo;
02 import jsystem.framework.system.SystemObjectImpl;
03 import com.aqua.sysobj.conn.CliCommand;
04 import com.aqua.sysobj.conn.CliConnectionImpl;
05 public class SimpleWindowsStation extends SystemObjectImpl {
06   public CliConnectionImpl cliConnection;
07   public void init() throws Exception {
08     super.init();
09     report.step("In init method");
10   }
11   public void close(){
12     report.step("In close method");
13     super.close();
14   }
15   public void mkdir(String folderName) throws Exception {
16     CliCommand cmd = new CliCommand("mkdir " + folderName);
17     cmd.addErrors("unknown command");
18     cliConnection.handleCliCommand("created dir " + folderName,
 cmd);
19     setTestAgainstObject(cliConnection.getTestAgainsObject());
20   }
21   public void dir(String folderName) throws Exception {
22     CliCommand cmd = new CliCommand("dir " + folderName);
23     cmd.addErrors("unknown command");
24     cliConnection.handleCliCommand("dir " + folderName, cmd);
25     setTestAgainstObject(cmd.getResult());
26   }
27   public void ping(String host) throws Exception {
28     CliCommand cmd = new CliCommand("ping " + host);
29     cmd.addErrors("unknown command");
30     cliConnection.handleCliCommand("ping " + host, cmd);
31     setTestAgainstObject(cliConnection.getTestAgainsObject());
32   }
33 }
```

Table 3: Cli Code Example

10.3.1.3 Cli SUT file for the SystemObject

Cli commands are written by creating a new instance of the "**CliCommand**" class, the Cli command is then executed by activating the "**handleCliCommand**" of the CliConnection entity.

```
01 <sut>
02   <station>
03     <class>com.aqua.services.demo.SimpleWindowsStation</class>
04     <cliConnection>
05       <class>com.aqua.sysobj.conn.WindowsDefaultCliConnection</
class>
06       <user edit="enable">simpleuser</user>
07       <password edit="enable">simpleuser</password>
08       <host edit="enable">127.0.0.1</host>
09       <connectOnInit>false</connectOnInit>
10     </cliConnection>
11   </station>
12 </sut>
```

Table 4: Cli SUT File Code Example

The "**CliConnection"** is a system object that can be initialized from a SUT file. The example illustrates how the DUT section appears.

Note: The cli Connection class is "com.aqua.sysobj.conn.WindowsDefaultCliConnection".

This class extends the "**com.aqua.sysobj.conn.CliConnectionImpl**" implementation and defines the prompts and configurations that are specific to the windows Telnet cli connection.

10.3.2 Ignis Stations Driver

10.3.2.1 Ignis Stations Driver Overview

Ignis's Stations driver is a family of System Objects specifically designed to expose a unified interface for remote controlling Windows and Linux machines.

The remote operations activation of the machines is performed by using the Cli driver, thus the stations system objects do not require the use of an agent and use standard protocols to perform their operations.

Once a Cli connection is made to the remote machine, standard applications can then be used to perform the communication operations.

Operations that are not supported on all operating systems require a specific implementation as per each operating system (for example, Windows registry operations).

10.3.2.2 Stations Specification Table

The following specification table details the Stations driver operating system specifications.

Specification	Description
Operating Systems	Windows 2000,XP,Vista Linux Ubuntu, Fedora, Debian
Pre Requisite	Cli agent (Telnet/SSh/RS232) has to be active on the controlled machine

Table 5: Station Driver Specifications

10.3.2.3 Stations Features

Function	Description
File Transfer	Transfers files from local machines to remote machines and vice versa.
	Works with Cli and FTP protocols. Does not require installation of FTT server.
Process Management and monitoring	Start/Kill process. Start process as user.
	Waits for process to end as well as analyzes the process exit code.
	Monitor process performance (CPU, memory, handles, thread).
	Gets process information (owner, used dll's/sos).
Services Management	Stop/Start/Restart service.
Network Adapter Management, Network Related Configurations	Set/Get network adapter IP protocol settings.
	Resets adapter.
	Gets general adapter settings.
	Hosts file manipulation.
File System	Analyzes file existence and properties.
	Compares files and folders.
	Generates files.
Batch Scripts Execution	Executes scripts in various scripting languages (batch scripts, TCL, Perl and Python).
General Operating System Operations	Map/Un map drives.
	Manipulates machines date/time.
	Manipulates users and groups.
	Restart/log-off machine.
Environment Variables	Sets and gets environment variables.
Common Applications	System object used for activating and analyzing Ping applications.

Table 6: CLI Features

10.3.2.4 Code Example

The code example illustrates the interfaces exposed by the “system objects” that manages the machine’s network adapter.

```
01 package com.aqua.station.windows.network;
02 import java.util.ArrayList;
03 /**
04  * Interface of system object for handling network adapter oper
ations05  *
06  */
07 public interface NetworkAtapter {
08   /**
09    * Resets network adapter.
10    * @param description -
 can be MAC address or other descriptive information
11    * @throws Exception
12    */
13   public void resetNetworkAdapter(String description) throws Ex
ception;
14
15   /**
16    * Sets network adapter to enabled/disabled state
17    * @param description
18    * @param state
19    * @throws Exception
20    */
21   public void setNetworkAdapterState(String description, Networ
kAdapterState state) throws Exception;
22   /**
23    * set static IP.
24    * @param interfaceName -
 The interface which its IP is going to be changed.
25    * @param ip - The new IP for the interface
26    * @param subnetMask - The subnt mask for this IP
27    * @throws Exception
28    */
29   public void setStaticIp(String interfaceName, String ip, Stri
ng subnetMask)
30       throws Exception;
31   /**
32    * Returns a list of the remote machine's interfaces
33    * It does not return the lo interface.
34    */
35   public ArrayList getInterfacesList() throws Exception;
36
37   /**
38    * Returns a list of the IPs of an interface
39    */
40   public ArrayList<IpStructure> getIps(String interfaceName) th
rows Exception;
41
42 }
```

Table 7: Network Adapter System object interface

10.3.3 Ignis Scripting Languages (Tcl) Driver

10.3.3.1 Tcl Scripting Languages Overview

Tcl (originally from "**Tool Command Language**", but nonetheless conventionally rendered as "Tcl" rather than "TCL"; pronounced as "tickle" is a scripting language. Originally, "born out of frustration" according to the author with programmers devising their own (poor quality) languages intended to be embedded into applications. It is most commonly used for rapid prototyping, scripted applications, GUIs and testing. Tcl is used extensively on embedded systems platforms, both in its full form and in several other small-foot printed versions.

10.3.3.2 Ignis Scripting Languages (Tcl) Driver Overview

One of the ways to interact with testing equipment is to activate the API that was written for it by the vendor. Usually the API of such tools is TCL or VB API.

Examples for such equipment and or applications are Ixia, Test Center and OmniPeek.

The scripting languages drivers enable the user to activate these APIs from the System Objects. The driver includes two drivers, one for interacting with Tcl scripts and the other to interact with VB Scripts.

Both drivers work in the same way: they spawn a new process which activates the interpreter of the scripting language, scripting commands are then passed to the interpreter from the input stream of the process, and the results of the command are then fetched from the output stream of the process and analyzed.

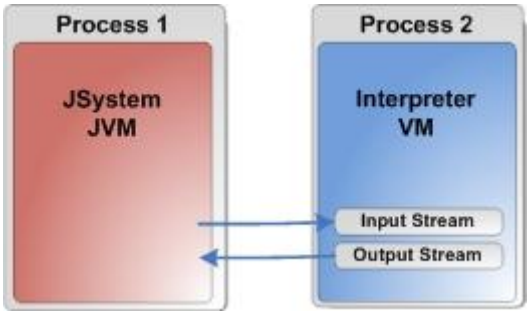

Figure 2: Scripting Language Diagram

JSystem spawns another process which runs the scripting language interpreter; it passes commands to the interpreter through process input stream and fetches the results of the command by reading data from process output stream.

The Scripting language driver is a low level driver so it is usually used by applicative drivers (system objects) and not directly by tests.

10.3.3.3 Driver Services

The driver services provided by this drive are as follows.

- Activation of interpreter.
- The path to the interpreter can be configured from SUT file.
- Easy to use API.
- Built-in analysis of commands results.
- Services for loading and interpreting scripts from java code.

10.3.3.4 Tcl Shell Code Examples

In the code example shows a JSystem test that activates a "**TclShell**" driver.

```
01 package tcl;
02 import java.io.File;
03 import java.io.IOException;
04 import junit.framework.SystemTestCase;
05 import com.aqua.tcl.ShellCommand;
06 import com.aqua.tcl.TclShell;
07 import com.aqua.tcl.TclShellLocal;
08 /**
09  */
10 public class TclExampleTest extends SystemTestCase {
11   TclShell shell;
12   public void setUp() throws Exception {
13     String shellPath = sut().getValue("/sut/tcl/shellPath/text(
)");
14     shell = new TclShellLocal(new File(shellPath));
15     shell.launch();
16   }
17   public void testShellCommad() throws IOException {
18     ShellCommand cmd = new ShellCommand();
19     cmd.setCommand("puts");
20     cmd.setParameters(new String[]{"Hello World 1"});
21     shell.executeCommand(cmd);
22     assertTrue("ShellCommand constructor with no parameters fai
led", cmd.getStdOut().indexOf("Hello World 1") >= 0);
23     report.report("ShellCommand constructor with no parameters
successfully completed");
24   }
25   public void tearDown(){
26     shell.exit();
27   }
28 }
```

Table 8: Ignis Scripting Language Code Example

In the setup method the test reads the path of the Tcl interpreter from the SUT file and creates an instance of the "**TclShellLocal**" class. This is the class that activates Tcl command and parses the results.

In the "**testShellCommand**" test case a Tcl command object is created and then executed.

Note: In addition to executing the command the "TclShellLocal" class also verifies that the command ended with success and that its error code is zero.

The teardown method shown in line 25 to 27disposes the Tcl shell object.

10.3.3.5 Code Example

The code shows a simple test that creates an instance of the "**VBShell**" system object, and sends a simple command; it then analyzes the command results and disposed the system object.

```
01 package com.aqua.vbshell;
02 import com.aqua.shell.common.ShellCommand;
03 import junit.framework.SystemTestCase;
04 public class VBShellTest extends SystemTestCase {
05   private VBShell shell;
06   public void setUp() throws Exception {
07     shell = (VBShell) system.getSystemObject("vbshell_1");
08   }
09   public void testInit() throws Exception {
10     shell.startInteractiveSession();
11     ShellCommand command = new ShellCommand("wscript.echo \"tes
t\"",new Object[]{});
12     shell.command(command);
13     assertEquals("test",command.getStdOut());
14     shell.close();
15   }
16 }
```

Table 9: Tcl Shell Object Code Example

The setup method provides an instance of the "**VBShell"** system object. The test method starts the session with the VB interpreter and sends an "**echo**" command to the test and then verifies that the text "**test**" was indeed echoed, it then closes the system object.

10.3.4 Ignis SNMP Driver

10.3.4.1 SNMP Protocol Overview

The Simple Network Management Protocol (SNMP) is essentially a request-reply protocol running over UDP (ports 161 and 162), though TCP operation is possible. SNMP is an asymmetric protocol, operating between a management station (smart) and an agent (dumb). The agent is the device being managed - all its software has to do is implement a few simple packet types and a generic get-or-set function on its MIB variables. The management station presents the user interface.

The following documents specify the Internet standards track protocols for the Internet community. For information and details concerning exact internet protocols refer to the following links:

http://www.ietf.org/rfc/rfc1157.txt

http://www.ietf.org/rfc/rfc1902.txt

http://www.ietf.org/rfc/rfc1903.txt

http://www.ietf.org/rfc/rfc1904.txt

http://www.ietf.org/rfc/rfc1905.txt

http://www.ietf.org/rfc/rfc1906.txt

http://www.ietf.org/rfc/rfc1907.txt

10.3.4.2 Ignis SNMP Driver Overview

The **Ignis SNMP Driver** is a "station" implementation. It provides both basic SNMP operations and advanced operations along with analysis capabilities:

Every time the SNMP is activated the **Ignis SNMP Driver** compiles the MIB files and saves the compilation products; it then enables dynamic binding of MIB OID using an MIB name during the test.

The **Ignis SNMP Driver** also adds a log description of every action made. The log contains the action type, MIB name, MIB OID, value, MIB declaration and description as they appear in the MIB file. In the event of a failure – reference to the log containing the exact failure reason as returned by the agent is possible.

Note: The SNMP driver can be initiated from the SUT file or dynamically during the test.

10.3.4.3 Ignis SNMP Driver Functionality Description

The functions supported by the Ignis SNMP driver are listed as follows:

Function	Description
Configuration	**SNMP Session Community** – Private/Public
	SNMP Version - SNMPv1/SNMPv2
	SNMP Specific Properties - Agent's IP,MIB's directory
Actions	**Get** – for a single object
	Get Next – for a single object
	Walk – On an entire MIBs table
	Set – for a single or multiple objects
	Complete Operation – List Of Set Actions Along With a List Of Mandatory And Non Mandatory Gets To Perform In Order To Analyze The Operation Results
	Trap Listening – A Listener for SNMPv1 and SNMPv2 traps. When it receives a trap – it adds the trap's details to its "traps table" which could be retrieved on any time and notify all its listeners
Analysis Results	Action Result Analyze.
	Returned Value Analyze

Table 10: Ignis SNMP Configuration Settings

10.3.4.4 SUT Code Example

The following code is an example of an SUT file that belongs to an SNMP driver.

```
<snmp>
<class>systemobject.snmp.Snmp</class>
<version>1</version> <!-- 0 for SNMPv1, 1 for SNMPv2 -->
<addLinkToReport>true</addLinkToReport> <!—log appearance -->
<community>private</community> <!-- "private" or "public" -->
<retries>5</retries> <!-- number of request retries -->
<timeout>10000</timeout> <!-- request timeout(milliseconds) -->
<port>161</port> <!-- UDP port, 161 or 162 -->
<host>1.2.3.4</host> <!-- IP of the agent/host -->
<mibsDir>c:\aqua\automation\mibFiles</mibsDir>
</snmp>
```

Table 11: SUT Example

10.3.4.5 Dynamic load

The following code example shows how to use the SNMP driver.

```
Snmp snmp = new Snmp();
snmp.setHost(host);
snmp.setVersion(SnmpSMI.SNMPV2);
snmp.setMibsDir(mibsDir);
snmp.loadMibsToMap();
```

Table 12: SUT Dynamic Code Example

10.3.4.6 Code Examples

The following code example illustrates advanced functionality and usage of the SNMP driver.

```
01 // walk with dynamic OID get from SNMP object
02 snmp.walk(snmp.getOidFromMap("exampleMibTable1"));
03 // walk with constant OID
04 snmp.walk("1.3.5.5.5.5.5.5.5");
05 //silent get-next with dynamic OID get from SNMP object
06 snmp.getNext(snmp.getOidFromMap("exampleMibName1")+"."+10000, t
rue);
07 // silent get-next with constant OID
08 snmp.getNext("1.3.5.5.5.5.5.5.5.5.10000", false);
09 //silent get with dynamic OID get from SNMP object
10 snmp.get(snmp.getOidFromMap("exampleMibName1")+"."+10000, true)
;
11 // silent get with constant OID
12 snmp.get("1.3.5.5.5.5.5.5.5.5.10000", false);
13 // perform analyze on the returned value
14 snmp.analyze(new CheckCounter(snmp.getLastOidUsed(), 1), false,
 false);
15 // get actual returned value as String
16 GetTextCounter tc;
17 tc = new GetTextCounter(snmp.getLastOidUsed());
18 snmp.analyze(tc);
19 String value = tc.getCounter().trim();
20 // single MIB set - for this example - Integer value "138"
21 snmp.set(new SnmpVarBind[]{new SnmpVarBind(
22 new SnmpObjectId(snmp.getOidFromMap("exampleMibName1")+"."+1000
0), new SnmpInt32(138)}, false));
23 // Multiple MIB set
24 SnmpVarBind first, second, third, forth;
25 first = new SnmpVarBind(
26      new SnmpObjectId(snmp.getOidFromMap("exampleMibName1")+"."
+10000),
27      new SnmpInt32(138));
28 second = new SnmpVarBind(
29      new SnmpObjectId(snmp.getOidFromMap("exampleMibName2")+"."
+10000),
30      new SnmpGauge32(120000L));
31 third = new SnmpVarBind(
32      new SnmpObjectId(snmp.getOidFromMap("exampleMibName3")+"."
+10000),
33      new SnmpOctetString(new byte[]{0x11, 0x80, 0x00}));
34 forth = new SnmpVarBind(
35      new SnmpObjectId(snmp.getOidFromMap("exampleMibName4")+"."
+10000),
36      new SnmpTimeTicks(150000L));
37 snmp.set(new SnmpVarBind[]{first, second, third, forth}, false)
);
38 // Multiple set actions using the SnmpSetAction Class With Sile
nt Validation
39 SnmpSetAction sa = new SnmpSetAction("Set Example");
```

```
40 sa.setSilentValidation(true);
41 sa.append(snmp.getOidFromMap("exampleMibName1")+"."+10000, 15);
42 sa.append(snmp.getOidFromMap("exampleMibName2")+"."+10000, 180000L);
43 sa.append(snmp.getOidFromMap("exampleMibName3")+"."+10000, "00:00:01");
44 sa.append(snmp.getOidFromMap("exampleMibName4")+"."+10000, new SnmpInt32(15));
45 sa.appendMandatoryValidate(snmp.getOidFromMap("result1")+"."+10000, new SnmpOctetString(new byte[]{0x00, 0x00, 0x01}));
46 sa.appendMandatoryValidate (snmp.getOidFromMap("result2")+"."+10000, new SnmpTimeTicks(180000L));
47 sa.appendNonMandatoryValidate (snmp.getOidFromMap("error1")+"."+10000, 1);
48 sa.appendNonMandatoryValidate (snmp.getOidFromMap("error2")+"."+10000, 1L);
49 snmp.set(sa);
```

Table 13: SNMP Code Example

Figure 3": SNMP

10.4 Network Drivers

The application drivers refer to the application discipline that enables the JSystem framework the ability to connect to test and network equipment providing testing functionality for testing devices.

The illustration shows the network application drivers within the system objects or drivers layer of the JSystem Framework.

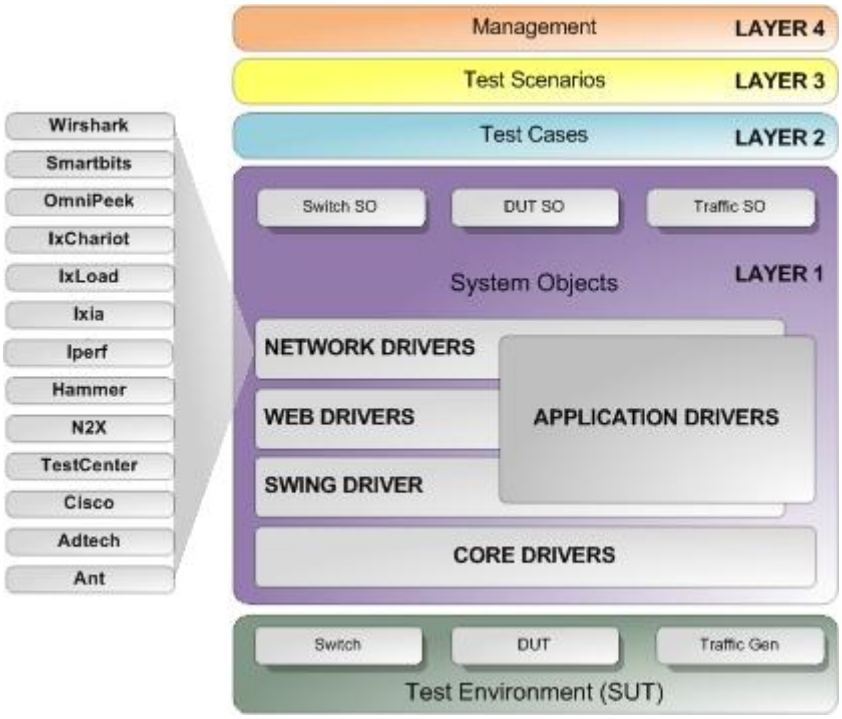

Figure 4: Network Application Drivers Layer

10.4.1 Ignis Wireshark Driver

10.4.1.1 Wireshark Overview

Wireshark is the world's foremost network protocol analyzer, and is the de facto standard across many industries and educational institutions.

10.4.1.2 Ignis Wireshark Driver Overview

The **Ignis Wireshark Driver** provides API to Wireshark functionality (this function operates via CLI mode only and controls the Tshark application). The **Ignis Wireshark Driver** provides an API to the Wireshark command line application (Tshark).

This application enables the user to, start and or stop sniffing, configure sniffing, and analyze Wireshark sniffing traffic results. The **Ignis Wireshark Driver** is interfaced to the Tshark program, the driver is saved on specific endpoints station using a JSystem CLI connection called, "CliConnection". The Wireshark driver controls and manages the Tshark program.

Note: The traffic results saved on the JSystem station running the tshark application does so in a cap file format, or directly as a message on the station monitor.

The **Ignis Wireshark Driver** can also filter saved cap files and create new ones that have a specific filter.

At the end of the Wireshark run the result files can be deleted from the station, zipped or they can be converted into text files. The cap files can be transferred to the JSystem JRunner station by using a FTP protocol to analyze the results.

10.4.1.3 Features

Wireshark has a rich feature set that includes functionality such as, deep inspection of hundreds of protocols. New protocols are constantly being added to the Wireshark database. The user can also perform live capture of the tests performed, and utilize the offline analysis function.

The captured network data can be browsed by using the Wireshark Tshark utility. Wireshark possesses the most powerful display filters in the industry. Rich VoIP analysis is also possible.

10.4.1.4 Wireshark Features

The following table provides a list of the file formats and technical functionality data.

Function	Description
Read/write: Capture File Formats	tcpdump (libpcap), Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network Monitor, Network General Sniffer® (compressed and uncompressed), Sniffer® Pro, and NetXray®, Network Instruments Observer, Novell LANalyzer, RADCOM WAN/LAN Analyzer, Shomiti/Finisar Surveyor, Tektronix K12xx, Visual Networks Visual UpTime, WildPackets EtherPeek/TokenPeek/AiroPeek.
Live Data Read from Ethernet	IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and others (depending on the user platform)
Decryption Support Protocols	IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2

Table 14: Wireshark Features

Note: Capture files compressed with gzip can be decompressed on the fly.

10.4.1.5 Wireshark Specification Table

The following specification table details the WireShark driver operating system specifications.

Specification	Description	
Supported Versions	0.99.4	
Operating Systems	Mac OSX	Solaris
	Vista, Windows XP	FreeBSD, NetBSD
	Linux	
Pre Requisite	Telnet agent on the machines on which Wireshark application is running.	
	Setting the tshark directory path in the System variables path.	

Table 15: Wireshark Application Specifications

10.4.1.6 Wireshark System Object (Driver) Operations

The following methods illustrate examples of the main methods used by a QA engineer.

Wireshark Entity Functionality	
Wireshark	init
	start
	stop
	filterCaptureFile
	filterCaptureFileAndSetTestAgainstObject
	getCaptureFile
	convertCapFileToTextFile
	restoreStartParams
	restoreStopParams
	restoreAllParams

Table 16: WireShark Method Examples

10.4.1.7 Code Example

The following code example demonstrates a 5 second activation of the Wireshark driver and filter of the capture file and assigning the result to the "**test against object**".

```
01 package com.aqua.examples.wireshark;
02 import com.aqua.wireshark.WireSharkManager;
03 import junit.framework.SystemTestCase;
04 public class WireSharkWithMatrix extends SystemTestCase {
05   private WireSharkManager manager;
06   public void setUp() throws Exception {
07     manager = (WireSharkManager)system.getSystemObject("wireSha
rkManager");
08   }
09   /**
10    * Kick start/Example for analysing cap file with PacketCaptu
re entity.
11    * In order for this example to work please copy
12    *  jpcap.dll to project folder.
13    */
14   public void testActivateWireSharkWithMatrix() throws Exceptio
n {
15     manager.wireSharkManagers[0].wireSharks[0].start();
16     sleep(5000);
17     manager.wireSharkManagers[0].wireSharks[0].stop();
18     manager.wireSharkManagers[0].wireSharks[0].setPacketTreeOut
put(true);
19     manager.wireSharkManagers[0].wireSharks[0].setReadFilter("
");
20     manager.wireSharkManagers[0].wireSharks[0].filterCaptureFil
eAndSetTestAgainstObject(manager.wireSharkManagers[0].wireSharks[0
].getCaptureFileName());
21
22   }
23 }
```

Table 17: Wireshark Code Example

10.4.2 Ignis SmartBits Driver

10.4.2.1 Smart Window Application Overview

The smart window application provides control functionality for managing Smartbits test hardware resources.

10.4.2.2 Ignis Smartbits Driver Overview

The **Ignis Smartbits Driver** enables full integration between the JSystem application and the manage Smart Window (Smartbits) chassis family providing take/release ownership and disconnect features.
Smartbits Driver Process Flow

The configuration, traffic and analysis functions supported by the Smartbits Ignis driver are listed as follows:

Specification	Description
Configuration	**Port Properties –** auto negotiation, MII, transmit modes
	Packets Streams – frame data, stream control
	Filters, Statistics and Receive Modes – triggers, filters, capture
Play Traffic	Start/Stop traffic play
	Pause/Resume traffic play
	Retrieve Data (statistics and packets) from Smartbits - ports/stream/flow statistics (counters and latency), captured frames
Analyze Results	Counters analyzers
	Frame analyzers

Table 18: Smartbits Ignis Driver Functionality

10.4.2.3 Traffic Play

The IxExplorer system object supports the following modes of play:

Specification	Description
Port Play	**Single Burst** - plays a burst of traffic and then stops and returns to the test.
	Multiple Bursts – plays a given burst several times
	Timed Play - plays traffic for a given period of time (e.g. 30 seconds)
	Continuous Play - Infinite play function. Starts playing the test and then returns immediately to the test, the test must stop the play.
Group Play ***(Define and play a group of ports)***	**Ports Burst** – different burst on each port in the group.
	Timed Play – plays all ports in a group for a given time.
	Continuous Play – play all ports infinitely till being stopped by the test.
	Step Play – play a single packet from each port in the group.

Table 19: Smartbits Port Configuration

Note: When play is stopped, either automatically or by the system objects or by a specific command from the test, the Smartbits system object retrieves the statistics and captured frames (if available) and then stores them for analysis.

During Timed play mode the test can ask the Smartbits system object to retrieve statistics for analysis.

10.4.2.4 Smartbits System Main Objects Component

The main class in the Smartbits system object is the Smartbits **com.aqua.traffic.smartbits.**Smartbits class;

The Smartbits class has an array of port classes, each representing a SmartBits port. Each Smartbits port has an array of **SmartBits Frams (SBFrame)** classes. Each port also holds an array of Streams it can play.

The various counter and frame analyzer classes are used to analyze the port counters, stream and flow counters and the captured frame.

10.4.2.5 Connectivity

The Smartbits system object connectivity to a Smartbits device is built over a Smartbits TCL API (SmartLib). To use this API the Smartbits system object launches a Tcl Wish interpreter and connects to its standard input, output and error channels.

To send a Tcl command the Smartbits system objects writes a command to the "wish standard input" so that the wish interpreter receives it as if it was entered via its shell window. Adversely, when the "wish interpreter" writes its output to the standard output or error, it reaches the Smartbits system object.

10.4.2.6 Validation

The Smartbits system object does not perform a validation for most parameters but relies on the **SmartLib** class to validate the user input.

Note: In cases where the **Smartbits TCL API** does not perform the required validation, the Smartbits system object validates the user input.

10.4.2.7 Adding a Tcl Script File to a Report

The Smartbits system object allows the user to save the created TCL script into a file so it can be used later for debug purposes.

If the user decided to save the TCL script into a file, the Smartbits system object will add a link to the file into the test report so that the user can view the script as part of the test report.

10.4.2.8 List of Smartbits Commands

The following table lists the Smartbits commands relevant to the JSystem Smartbits Driver.

Entity	Family	Functionality	Options
Smartbits	Management	Connect	Device ip
		Login	User name
		Disconnect	
		Retrieve Counter	For all reserved ports, both total and rate
		Play	Ports group
Port	Management	Ownership	Take/Release
		Reset	Reset to factory default
			Reset counters
	Properties	Auto Neg	Enable/Disable
			10/100/1000
			Half/Full
		Transmit Mode	Port play/Smart Metrics (Streams)
		MII Registers	Read/Write
		Retrieve Counters	For the port
		Port Utilization	Set the Utilization for port play
		Receive Trigger	Offset, range and value
SBframe	Frame Data	Frame size	Fixed
			Random Size
		Pause frames	Set frames to pause other ports
		Errors	No/Alignment/Dribble/Bad CRC/No CRC
		DA/SA	Mode (all)
			MAC address
			Repeat count
			Step
		VLAN (Single/Stacked)	VLAN ID
			User priority
			CFI
			Mode (Fixed/Incr/Decr)
			Repeat Count
			Step
		MPLS (up to 8 labels)	Label

			Priority
			TTL
		L3	None
			IPv4
			IPv6
			ARP
			IPX
		L4	TCP
			UDP
			ICMP
			IGMP
			DHCP
		IP Header V4	Source/Destination IP Address
			Source/Destination subnet mask
			Mode (all)
			Repeat count
			Step
			TOS Bits
			Fragmentation (bits+offset)
			TTL
		IP Header V6	Source/Destination IP Address
			Traffic class
			Flow label
			Hop limit
		TCP	Source/Destination port
		UDP	Source/Destination port
Smartbits Stream	Streams	DA\SA	
		Vlan	Priority,cif
		ToS	
Statistics	Port	Total/Rate	Packets Sent
			Bytes Sent
			Bits Sent
			Packets Received
			Bytes Received
			Bits Received
			FCS Errors
			Oversize
			Oversize and CRC
			Undersize
			Fragments
			Collisions
			Collision Frames
			Late Collisions
			Alignment Errors
			Frame Received
		Latency	Min

			Max
			Average
	Stream	Total	Frames Sent
		Rate	Frames Sent
Capture	Frames		

Table 20: Smartbits Commands

10.4.2.9 Code Example

The following code example defines ports for receiving and transmitting traffic, configures frame data, plays traffic and analyzes the results.

```
01 package com.aqua.examples.smartbits;
02 import junit.framework.SystemTestCase;
03 import systemobject.aqua.traffic.Frame;
04 import systemobject.aqua.traffic.TrafficUtil;
05 import systemobject.aqua.traffic.analizers.FrameField;
06 import systemobject.aqua.traffic.analizers.FrameFieldFilter;
07 import systemobject.aqua.traffic.analizers.MultiFrameCapture;
08 import com.aqua.traffic.smartbits.Smartbits;
09 import com.aqua.traffic.smartbits.SmartbitsPort.PortDuplex;
10 import com.aqua.traffic.smartbits.SmartbitsPort.PortSpeed;
11 /**
12  * test Ports configuration
13  *
14  * @author nizanf
15  *
16  */
17 public class ExampleTest extends SystemTestCase {
18   Smartbits sb;
19   public void setUp() throws Exception{
20     sb = (Smartbits)system.getSystemObject("sb");
21     sb.setPlayers(new int[]{0,1});
22     sb.setReceivers(new int[]{0,1});
23   }
24   /**
25    * test that the type tag is changed ok
26    *
27    * @throws Exception
28    */
29   public void testTypeTag() throws Exception{
30     sb.ports[0].resetPort();
31     sb.ports[0].frame.setFrameLength(60);
32   sb.ports[0].frame.setDaSa("00:00:11:22:33:44","00:00:11:22:33
:55");
33     sb.ports[0].setRate(1);
34     sb.ports[0].setDuplex(PortDuplex.FULL_DUPLEX);
35     sb.ports[0].setSpeed(PortSpeed.SPEED_100MHZ);
36     sb.ports[0].setBurstSize(100);
37     sb.setPlayers(new int[]{0});
38     sb.setReceivers(new int[]{1});
39     sb.ports[1].startCapturing();
40     String[] Addresses = new String[]{"00:00:00:00:00:01","00:0
```

```
0:00:00:00:02","00:00:00:00:00:03","00:00:00:00:00:04"};
41      int[] types = new int[]{0x9234,0x8a88,0xABCD,0xFFFF};
42      for (int i=0 ; i<Addresses.length ; i++){
43        sb.ports[0].frame.setVlan1(types[i],0, 0,0);
44        sb.ports[0].frame.setDAMac(Addresses[i]);
45        sb.playStep(true, true, false);
46      }
47      sleep(3000);
48      sb.ports[1].stopCapture(true);
49
50      for (int i=0 ; i<Addresses.length ; i++){
51        sb.ports[1].analyze(new MultiFrameCapture(
52            new FrameFieldFilter[]{new FrameFieldFilter(Frame.FIE
LD_DA_MAC,Addresses[i])},
53            new FrameField[]{new FrameField(Frame.FIELD_TYPE,Traf
ficUtil.convertLongToByteArray(types[i], 2))},1,475));
54      }
55    }
56 }
```

TaTable 21: Smart Table 22: Smartbits Code Example

10.4.3 Ignis OmniPeek Driver

10.4.3.1 OmniPeek Overview

The Omnipeek driver provides dual functionality as both a portable network analysis solution and as a software console for OmniEngines, OmniPeek offers an intuitive, easy-to-use graphical interface that engineers can use to rapidly analyze and troubleshoot enterprise networks. OmniPeek provides centralized expert analysis for all networks under management. With OmniPeek, Enterprise IT professionals can also analyze local segments on a portable basis. Using Omnipeeks intuitive user interface and "top-down" approach to visualizing network conditions, network engineers – even junior staff – can quickly analyze faults from multiple network segments, drill down through multiple layers of analysis, and pinpoint problems that need correction.

10.4.3.2 Ignis OmniPeek Driver Overview

The **Ignis Omnipeek Driver** has the ability to load an Omnipeeks configuration file and signal the Omnipeek application to start and or stop sniffing the network adapter.

Once the sniffing is completed the user can then fetch captured traffic in several formats (HTML, XMLK, and CAP) and then analyze the traffic.

The Omnipeek application exposes a COM API which enables any external application to activate the Omnipeek UI and start probing network interfaces.

Omnipeeks COM API is accessed by the **Ignis Omnipeek Driver** using VBScript scripts. These scripts are activated on the machine installed with Omnipeek using a telnet session.

10.4.3.3 OmniPeek Specification Table

The following table lists the Omnipeek application functional specifications.

Specification	Description
Supported Versions	Aeropeek 3.1
	OmniPeek 5.0
Operating Systems	Windows XP (and other windows versions that Omnipeek application runs on)
Pre Requisite	Telnet agent on the machine on which Omnipeek application is running

Table 23: OmniPeek Application Specifications

10.4.3.4 Omnipeek System Object Operations

The following table shows the list of supported Omnipeek operations.

Omnipeek Entity	
AiroPeek	startAiroPeek
	loadConfigurationFile
	startCapture
	stopCapture
	getCapture

Table 24: OmniPeek System Object Operations

10.4.3.5 Code Example

The following code example shows illustrates the uploading of a configuration, the activation of the Omnipeek for 60 seconds and the transfer of filter traffic.

```
01 InputStream stream = getClass().getResourceAsStream("NoBeacon.
ctf");
02 airopeek.startAiropeek();
03 airopeek.loadConfigurationFile(stream);
04 airopeek.startCapture();
05 Thread.sleep(1000*60);
06 airopeek.stopCapture();
07 File airopeekCapture = new File("resCapture."+Airopeek.HTML_DE
CODED_PACKET_FILE.getFileExtension());
08 airopeek.getCapture(airopeekCapture, Airopeek.HTML_DECODED_PAC
KET_FILE);
```

Table 25: OmniPeek Code Example

10.4.4 Ignis IxChariot Driver

10.4.4.1 IxChariot Overview

IxChariot is a traffic generation and decision support application that enables the user to emulate real-world application data without the need to install and maintain extensive client/server networks.

IxChariot provides thorough performance assessment and device testing by emulating hundreds of protocols and applications across thousands of network endpoints. Available with both node-locked and floating license support, IxChariot provides the ability to confidently predict the expected performance characteristics of any application running on wired and wireless networks. Using application scripts that emulate application data flows, IxChariot can help you:

- Test the performance and capacity of network hardware and software.
- Compare competing network products before purchase.
- Identify the source of performance problems.
- Predict the effects of running new applications.
- Measure and baseline typical network operations.
- Verify the performance that is expected from the network service providers.

IxChariot consists of a Console program and distributed Performance Endpoint programs. Using flows of real data, IxChariot emulates different kinds of distributed applications, and captures and analyzes the resulting performance data. Sample IxChariot tests are available from the Ixia Web site at www.ixiacom.com/support/IxChariot/ .

10.4.4.2 Single Network Test Environment

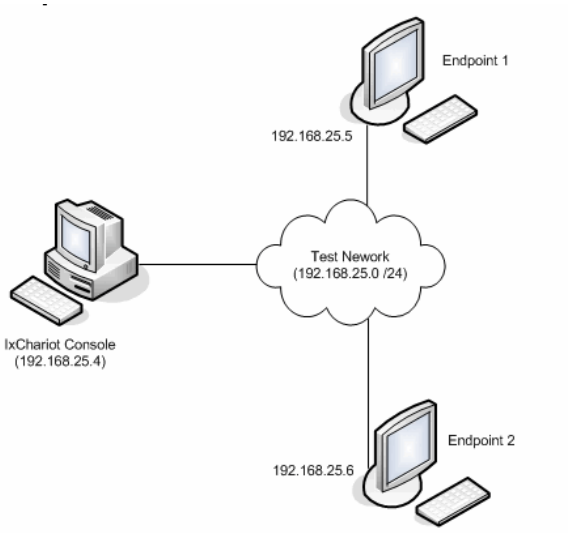

Figure 5: Single Network Test

Another variation of two-network test environment is shown in the illustration below. This configuration logically isolates the management network from the test network. The management network includes the IxChariot Console workstation (192.168.25.4) and the management addresses associated with each endpoint (192.168.25.5 and 192.168.25.6). The endpoints also have interfaces on the test network (network 172.16.1.0/24 in this example).

10.4.4.3 Two Network Test Environment

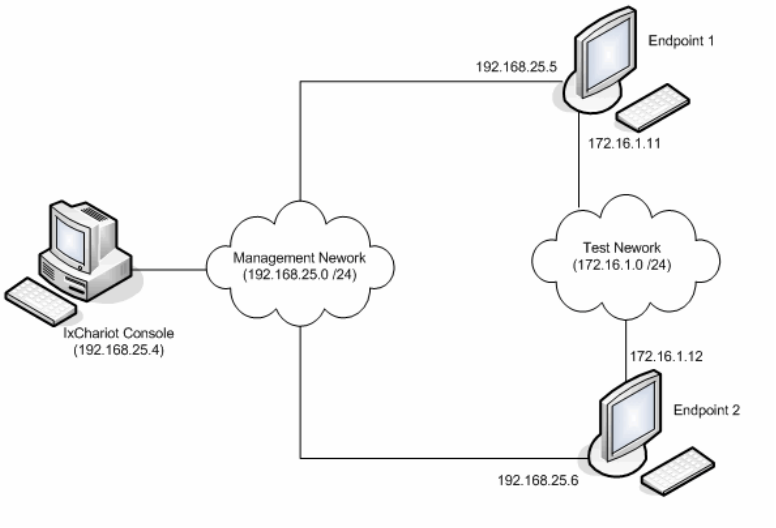

Figure 6: Multiple Network Test Environments

10.4.4.4 Ignis IxChariot Driver Overview

The **Ignis IxChariot Driver** provides API to IxChariot functionality. The IxChariot Ignis driver provides an API with parallel functionality to the IxChariot TST GUI file. This enables the user to configure, run, read statistics and analyze counters and packets.

The **Ignis IxChariot Driver** is interfaced to IxChariot by the TCL scripting language. The IxChariot API lets the user write TCL commands (Using java) that drive the IxChariot engine. The TCL commands write over the java code and use the API or the TCL package to interact with the IxChariot engine that contains the code that runs IxChariot tests and collects the results. By using the IxChariot API, the user can configure tests, run tests, save tests, and extract results.

There are two IxChariot consoles:

- **Light version –** Using this version the user can configure only IP addresses.
- **Full version –** Using this version provides the ability to configure every parameter the IxChariot test.

When using the **Ignis IxChariot Driver** on a station that has chariot light installed, the user can only use the java methods that are supported by IxChariot light. If the full IxChariot application is installed on the machine all Java methods are supported.

10.4.4.5 IxChariot Specification Table

The following table lists the IxChariot application functional specifications.

Specification	Description
Supported Versions	6.50
	6.4.0
Operating Systems	Windows XP
Pre Requisite	Telnet agent on the machine on which IxChariot endpoint application is running, IxChariot EndPoint, TCL SHELL on the machine that own the IxChariot console.

Table 26: IxChariot Application Specifications

10.4.4.6 IxChariot System Object Operations

The following table details the functions that are supported by the IxChariot driver.

IxChariot Functional Entities	
init	getDatagramsLost
transferTstFilesToChariotMachine	getDupPacksRecvByE1
transferTstFileToChariotMachine	getDupPacksRecvByE2
getResultsFileFromChariotMachine	getDupPacksSentByE1
setTest	getDupPacksSentByE2
saveTest	getOutOfOrderPackets
loadTest	getMediaLossRate
setTestEndCondition	getDelayFactor
setTestDuration	getMeanOpinionScore
runniest	getJitter
stopTest	getThroughput
createNewPair	getOneWayDelay
getPairCount	getRunStatus
getPairHandler	analyze
lockPair	close
unlockPair	getTst_location
setE1ManagementAddress	setTst_location
setE2ManagementAddress	getResult_location
getE2ManagementAddress	setResult_location
getE1ManagementAddress	IxChariot Full Version Methods
setE1_E2addresses	changeSendDataRate
setRunOpt	changeSendBufferSize
getRunOptsHandle	changeReceiveBufferSize
isTestRun	changeFileSize
waitForTestStop	changeVideoTimingRecordsDuration
waitForPairsToStopGracefully	changeVideoBitRate
getTestCompletionStatus	copyPair
getRunTime	copyVideoPair
useScript	
setScriptVar	
exit	
getPacketsSentByE1	
getPacketsReceivedByE2	
getPacketsReceivedByE1	
getDatagramsSent	

Table 27: IxChariot System Object Operations

10.4.4.7 Code Examples

The following code example shows a one “pair” traffic generation and its subsequent analysis.

```
01 import java.util.HashMap;
02 import java.util.Map;
03 import com.aqua.ixchariot.AvgMinMaxAnalyzer;
04 import com.aqua.ixchariot.IxChariot;
05 import com.aqua.ixchariot.RunOpt;
06 import jsystem.extensions.analyzers.text.CheckTextCounter;
07 import jsystem.extensions.analyzers.text.GetTextCounter;
08 import jsystem.extensions.analyzers.text.TextNotFound;
09 import junit.framework.SystemTestCase;
10 public class StationBasicTrafficTest extends SystemTestCase {
11   private IxChariot chariot;
12   private String chariot_TEST = null;
13   private String E1_MNG_ADDRESS = null;
14   private String E2_MNG_ADDREESS = null;
15   public void setUp() throws Exception {
16     chariot = (IxChariot)system.getSystemObject("ixchariot");
17   }
18   public void testStationBasicTraffic() throws Exception {
19     chariot.setTest( chariot_TEST );
20     chariot.loadTest();
21     chariot.clearResults();
22     int pairHandler = chariot.getPairHandler(0);
23
24     chariot.setE1ManagementAddress( pairHandler, E1_MNG_ADDRESS );
25     chariot.setE2ManagementAddress( pairHandler, E2_MNG_ADDREESS );
26     chariot.runTest();
27     chariot.waitForTestStop(310);
28     report.report( "Throughput : "+chariot.getThroughput(pairHandler) );
29     try {
30       chariot.analyze( new AvgMinMaxAnalyzer( "Throughput : ", EXP_THR,
ThroughputTolerance ) );
31     }
32     catch ( Exception e ) {
33       report.report(e.getMessage(), false);
34     }
35     report.report( "Save test results." );
36     chariot.saveTest();
37     report.addLink("IxChariot test results", "\\\\nt-ws-
stream41\\WORK\\Tests\\AQUA_DEMO\\DS_1x30Mbps_1.tst" );
38   }
39
40   double LossTolerance = 0.005;
41 }}
```

Table 28: IxChariot Code Example

10.4.5 Ignis IxLoad Driver

10.4.5.1 IxLoad Overview

Aptixia IxLoad is a highly scalable, integrated test solution for assessing the performance of Triple Play networks and devices. IxLoad emulates IPTV and Triple Play subscribers and associated protocols to ensure subscriber Quality of Experience (QoE). Protocols supported include video protocols like IGMP, MLD, and RTSP; voice protocols like SIP and MGCP; and data protocols like HTTP, FTP, and SMTP. In addition, IxLoad can be used to test critical aspects of the infrastructure like DNS, DHCP, RADIUS, and LDAP services, as well generate malicious traffic to test for security.

10.4.5.2 Ignis IxLoad Driver Overview

The **Ignis IxLoad Driver** is a relatively simple driver with very strong capabilities. The driver knows how to load an IxLoad repository file, select one or more tests and activate the test.

The **Ignis IxLoad Driver** works with an entity called "repository" a repository (.rxf file) includes in it one or more tests and each test has it's configuration which includes one or more chassis, ports mapping, ramp-up time etc'.
The IxLoad driver knows how to activate tests in a repository gather execution statistics and generate detailed reports.

The user can create **any** test with any protocol using IxLoad application and run them using Ignis's driver.

IxLoad functionality is activated by activating IxLoad's TCL APIs from Ignis's driver java code.

10.4.5.3 IxLoad Specification Table

The following table lists the IxLoad application functional specifications.

Specification	Description
IxLoad Version	3.x
Operating Systems	Windows XP

Table 29: IxLoad Application Specifications

10.4.5.4 IxLoad System Main Object and Operations

Working with the IxLoad drivers involves one main object the IxLoad object which exposes the following operations:

10.4.5.5 Code Example

The following code shows the interfaces of the IxLoad driver.

```
01 /**
02 * Sets the active tests repository
03 */
04 public synchronized void setActiveRepository(String repository)
05 /**
06 * sets the active test
07 */
08 public synchronized void setActiveTest(String testName)
09 /**
10 */
11 public synchronized void runTest() throws Exception;
12 /**
13  * Stops test's execution, cleans resources and generates repor
ts
14 */
15 public synchronized void stopTest() throws Exception
16 /**
17 * Returns true if a test is currently running, otherwise return
s false.
18  */
19 public synchronized boolean isTestRunning() throws Exception
20 /**
21 * Waits for test to end for <code>timeout</code>
22 * If test has ended on time, test's resources are cleaned, repo
rts + link in the
23 report are generated
24 * (if configured) and the method returns true.
25 * Otherwise, test continues to run and the method returns false
.
26 */
27 public synchronized boolean waitForTestEnd(long timeout) throws
 Exception
28 /**
29 */
30 public String getTestResultsFolder() throws Exception
```

Table 30: IxLoad Code Example

10.4.5.6 Analyzers

Analysis of IxLoad results is done using Ignis's CSV analyzers.

10.4.5.7 Code Example

The following code example demonstrates the loading of an IxLoad repository file, runs the test and analyzes the results.

```
01 public void testRunTestNonePendingNativeWait() throws Exception
 {
02 ixLoad.setActiveRepository("QuickTCP_CC_10GE-LSM.rxf");
03 ixLoad.setPending(false);
04 ixLoad.setGenerateReports(true);
05 ixLoad.runTest("Test1");
06 ixLoad.waitForTestEndNative();
07 validateResult();
08 }
```

Table 31: IxLoad Code Example 2

10.4.6 Ignis Ixia Driver

10.4.6.1 IxExplorer Application Overview

The IxExplorer application provides control functionality for managing Ixia test hardware resources. Complete control is provided for generation and analysis of Layer 2-4 traffic streams on an array of network interface technologies, including Ethernet,10 Gigabit Ethernet, Packet over SONET (POS), ATM, Frame Relay, etc. Ixia test ports can be independently configured to define traffic, filtering, and capture capabilities. Comprehensive statistics and graphical views enable in-depth analysis of the performance and functionality of the Device Under Test (DUT).

10.4.6.2 Ignis IxExplorer Driver Overview

The **Ignis IxExplorer (IXIA) Driver** provides API to Ixia functionality. The **Ignis Ixia Driver** provides an API with parallel functionality to the IxExplorer GUI application. This enables the user to configure, run, read statistics and analyze counters and packets. The **Ignis Ixia Driver** is interfaced with Ixia chassis family using Ixia's Tcl API activated by the Ignis Ixia driver.

10.4.6.3 IxExplorer Specification Table

The following table lists the IxExplorer application functional specifications.

Specification	Description
Application Type	Client application executable on PCs and workstations running Microsoft Windows operating systems
Platforms	Optixia® XL10, Optixia® X16, IXIA1600T, IXIA 400T, IXIA 250
Operating Systems	Microsoft Windows NT, 2000, XP
Chassis Management	Displays and provides control for multiple chassis chains with up to 256 chassis in each chain
Traffic Statistics	Flexible statistics display views in table and graph formats
Streams	Fine tune control of stream parameters such as frame size, data patterns, protocols, and rates
Filters	Control of traffic filtering parameters, such as addresses, custom data patterns, errors, and frame sizes
Multi-User	Login/logout for multi-user management of individual ports
Protocols and Encapsulations	IPv4, IPv6, TCP, UDP, IPX, VLAN, QinQ, Cisco ISL, ARP, ICMP, IGMPv1/v2/v3, MPLS, GRE, OSPF, RIP, DHCP

Table 32: IxExplorer Application Specifications

10.4.6.4 Ignis Ixia Driver Functionality Description

The Ixia system object (Ignis driver) enables full integration between the JSystem Automation Platform and the manage IxExplorer (IXIA) chassis family providing take/release ownership and disconnect features.

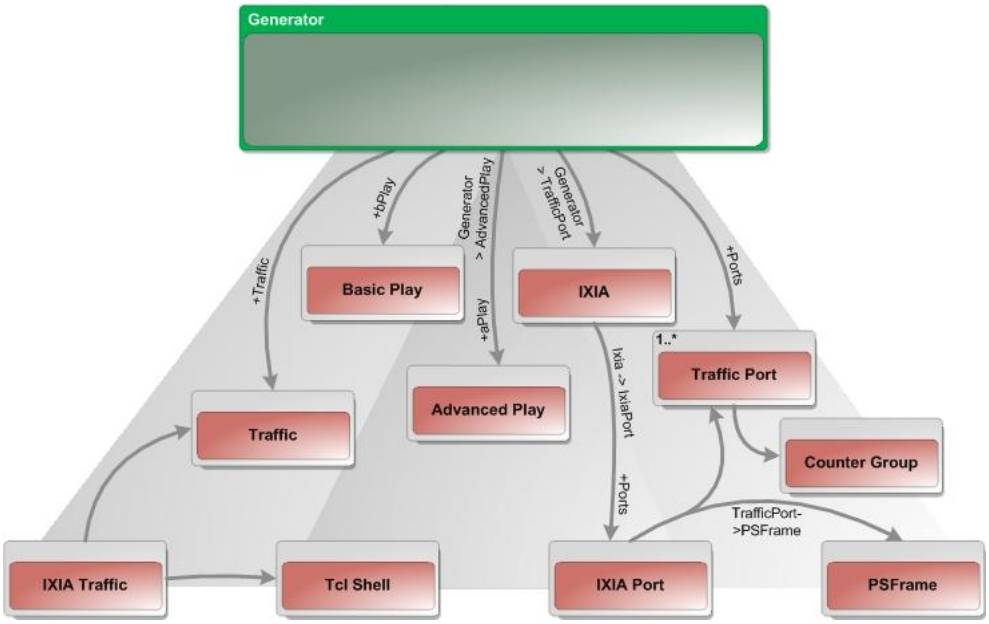

Figure 7: Ixia Driver Process Flow

The configuration, traffic and analysis functions supported by the IxExplorer Ignis driver are listed as follows:

1. **Configuration:**
   - **Port Properties** – auto negotiation, MII, transmit modes
   - **Packets Streams** – frame data, stream control
   - Filters, Statistics and Receive Modes – triggers, filters, capture
1. **Play Traffic:**
   - Start/Stop traffic play
   - Pause/Resume traffic play
   - **Retrieve Data (statistics and packets) from Ixia** – ports/stream/flow statistics (counters and latency), captured frames
2. **Analyze Results:**
   - Counters analyzers
   - Frame analyzers

10.4.6.5 Chassis Management

To view the complete list of the IxExplorer system object management functionality supported by the IxExplorer Ignis driver, refer to the IxExplorer commands Table.

10.4.6.6 Configuration

To view the complete list of the IxExplorer system object configuration functionality supported by the IxExplorer Ignis driver, refer to the IxExplorer commands Table.

10.4.6.7 Traffic Play

The IxExplorer system object supports the following modes of play:

1. **Burst Play** – plays a burst of traffic and then stops and returns to the test.
2. **Timed Play** – plays traffic for a given period of time (e.g. 30 seconds) and then stops and returns to the test.
3. **Continuous Play** – Infinite play function. Starts playing the test and then returns immediately to the test, the test must stop the play.

Note: When play is stopped, either automatically or by the driver (system object) or by a specific command from the test, the IxExplorer system object retrieves the statistics and captured frames (if available) and then stores them for analysis.

During Timed play mode the test can ask the IxExplorer system object to retrieve statistics for analysis.

10.4.6.8 Learning

The IxExplorer system object supports a learning function before playing the test. This function sets the IxExplorer to send a 10 packet burst enabling the DUT to learn the MAC addresses before playing the actual test.

10.4.6.9 Router

The IxExplorer system object supports sending ARP's before playing the test. This function sets the IxExplorer to send ARP requests to resolve the IP address of the neighboring device. Using this function the test can run dynamically with router ports.

10.4.6.10 Host

The IxExplorer system object supports setting the IxExplorer to send/answer ping requests. This function sets the IxExplorer to act as a host rather than a traffic generator.

10.4.6.11 Ixia System Main Objects Component

The main class in the Ixia system object is the Ixia "**com.aqua.traffic.ixia.Ixia"** class; this represents the chassis. The Ixia class has an array of port classes, each representing an Ixia port. Each Ixia port has an array of **Packet Stream Frame (PSFrame)** classes. Each port also holds a class to store the stream counters.

The Ixia traffic "**com.aqua.traffic.ixia.Ixia"** class also has **play** class **"systemobject.aqua.traffic.AdvancedPlay"** responsible for all play operations. The **play** class holds the counters for all ports that participate in the play test.

The "**IxiaTraffic**" "**com.aqua.traffic.ixia.IxiaTraffic"** class is the class that connects to the IxExplorer device. All classes that need to access the Ixia device use it.

The various counter and frame analyzer classes are used to analyze the port counters, stream and flow counters and the captured frame.

10.4.6.12 Connectivity

The IxExplorer system object connectivity to an IxExplorer device is built over an Ixia TCL API (**IxTclHal**). To use this API the IxExplorer system object launches a Tcl Wish interpreter and connects to its standard input, output and error channels.

To send a Tcl command the Ixia system objects writes a command to the "wish **standard input**" so that the wish interpreter receives it as if it was entered via its shell window. Adversely, when the "**wish interpreter**" writes its output to the standard output or error, it reaches the IxExplorer system object.

10.4.6.13 Validation

The IxExplorer system object does not perform a validation for most parameters but relies on the "**IxiaTclHal"** class to validate the user input.

Note: In cases where the Ixia Tcl API does not perform the required validation, the IxExplorer system object validates the user input.

10.4.6.14 Adding a Tcl Script File to a Report

The IxExplorer system object allows the user to save the created Tcl script into a file so it can be used later for debug purposes.

If the user decided to save the Tcl script into a file, the IxExplorer system object will add a link to the file into the test report so that the user can view the script as part of the test report.

10.4.6.15 Analyzers

10.4.6.16 Ethernet Counters

The Ixia drive supports three types of counters – rate, total and latency counters. These counters are supported per port as well as per flow.

Note: In order to see per stream counters in the IxExplorer GUI interface the user must map each stream to a flow and then read the flow counters.

Ixia driver allows the user to map streams, MAC addresses and VLANs to flows and read statistics so the user can analyze per stream, per MAC and per VLAN statistics.

To view the complete list of the IxExplorer counters supported by the IxExplorer Ignis driver, refer to the IxExplorer commands Table.

10.4.6.16.1 Ethernet Counters Analyzers

There are two basic types of counter analyzers – static and dynamic. The static analyzer compares a counter to an expected given value. The dynamic analyzer compares a counter to an expected value dynamically extract from another counter.

Note: Both analyzers verify that the counter is in within range of an expected value. That is, Expected – Tolerance <= Counter <= Expected – Tolerance. This tolerance can be set as a percentage or an absolute number.

There are many derivative analyzers from these two basic types.

For example, a counter that compares received packets on one port to sent packets from another port.

10.4.6.16.2 Packet Analyzers

There are two groups of packet analyzers, field and frame.

There are two types of field analyzers:

- **Field** – this is an analyzer that compares a specific field within the packet (DA MAC, source IP etc.) to an expected value. An expected value can be a string, integer or an int[] based on the field type.
- **Offset** – this is an analyzer that compares an absolute offset within the packet to an expected value given as int[].

There are two types of frame analyzers:

- **Whole** – this is an analyzer that compares a frame to an expected frame.
- **Multi Fields** – this is an analyzer that compares a set of fields within the frame to a set of fields within a given frame.

10.4.6.17 List of IxExplorer Commands

The following table lists the functionality and options in the IxExplorer.

Entity	Family	Functionality	Options
Chassis	Management	Connect	Chassis ID
		Login	User name
		Disconnect	
Port	Management	Ownership	Take/Release
		Reset	Reset to factory default
			Reset counters
	Properties	Auto Neg	Enable/Disable
			10/100/1000
			Half/Full
		Transmit Mode	Packet Stream/Advanced Stream
		PHY Modes	Copper/Fiber
		MII Registers	Read/Write
		State	Up/Down
		Link Fault Signaling	Error blocks
			Good blocks
			Loop count/forever
			Ordered sets (type A / type B / Alternate
			Enable/Disable ignore Rx link faults
			Ordered set type A (local/remote/custom)
			Ordered set type B (local/remote/custom)
		Xaui	Clock type (Internal/external)
		Flow Control	Directed address
			Multicast address
		Xenpak	Enable/Disable LASI monitoring
			Enable/Disable Auto Detect OUI device address
			Manual OUI device address
			Rx alarm control register
			Tx alarm control register
			LASI control register
		Preamble	Tx Mode (SFD/Byte)
			Rx Mode (Same/SFD/Byte)
			Enable/Disable Cisco CDL
			Enable/Disable view preamble
			Enable/Disable CDL statistics
		Pre emphasis	

		Clock deviation	
Packet Stream	Frame Data	Frame size	Fixed
			Uniform Random (min, max)
			Weighted Random (size, rate)
			Increment (step, min, max)
			Auto
		Insert modes	Timestamp
			Packet group signature (+signature offset + group ID offset)
			Data integrity (+offset)
			Sequence number
		Errors	No/Alignment/Dribble/Bad CRC/No CRC
		DA/SA	Mode (all)
			MAC address
			Repeat count
			Step
		UDF (1-4)	Mode (Up Down Counter/Random)
			Type
			Offset
			Initial value
	Protocols	DLL	None
			Ethernet II (+ type)
			Ethernet Snap
			802.3 Raw
			802.2 IPX
			Protocol offset + user defined protocol
		VLAN (Single/Stacked)	VLAN ID
			User priority
			CFI
			Mode (Fixed/Incr/Decr)
			Repeat Count
			Step
		MPLS (up to 8 labels)	Label
			Priority
			TTL
		L3	None
			IPv4
			IPv6
			ARP
			IPX
		L4	TCP
			UDP

			ICMP
			IGMP
			DHCP
		ARP	Operation (all)
			Sender HW address (+mode+count)
			Sender protocol address (+mode+count)
			Target HW address (+mode+count)
			Target protocol address (+mode+count)
		IP Header V4	Source/Destination IP Address
			Source/Destination subnet mask
			Mode (all)
			Repeat count
			Step
			TOS Bits
			Fragmentation (bits+offset)
			TTL
		IP Header V6	Source/Destination IP Address
			Traffic class
			Flow label
			Hop limit
		TCP	Source/Destination port
		UDP	Source/Destination port
	Stream Control	Rate	Percentage
			PPS
			BPS
			IPG in bytes
		Navigation	Continues Packet
			Stop after this Stream (+Packets Per Burst)
			Advance to Next Stream (+Packets Per Burst)
			Return to ID (+Packets Per Burst+ ID)
			Return to ID (+Packets Per Burst+ ID+loop)
Filters	Filter Properties	General	Enable/Disable (1-6)
			SA Mode (all)
			DA Mode (all)
			Pattern Mode (all)
		DA/SA	Address
			Mask
		Pattern1	Offset
			Pattern
			Mask

		Pattern2	Offset
			Pattern
			Mask
	Receive Mode	Mode	Capture/Wide Packet Groups/Data Integrity
		(Wide) Packet Groups	Signature offset
			Group ID offset
			Group ID mask
		Data Integrity	Data Integrity offset
Protocol Window	Protocol Management	Enable ARP	Enable
		Enable Ping for IPv4	Enable
		Protocol ifs table	IPv4 Address
			IPv4 Mask
			IPv4 Gateway
		IP table	IPv4 Address
			IPv4 Mask
			IPv4 Gateway
Statistics	Port	Total/Rate	Packets Sent
			Bytes Sent
			Bits Sent
			Packets Received
			Bytes Received
			Bits Received
			FCS Errors
			Oversize
			Oversize and CRC
			Undersize
			Fragments
			Collisions
			Collision Frames
			Late Collisions
			Alignment Errors
			customOrderedSetsReceived
			customOrderedSetsSent
			localFaults
			localOrderedSetsReceived
			localOrderedSetsSent
			remoteFaults
			remoteOrderedSetsReceived
			remoteOrderedSetsSent
		Latency	Min
			Max
			Average
		Filters and	Data Integrity Errors

		Triggers	
			Data Integrity Frames
			User Defined 1
			User Defined 2
			Capture Trigger
			Capture Filter
			Stream Trigger 1
			Stream Trigger 2
	Flow	Total	Packets Received
		Rate	Bit Received
			Byte Received
			Frame Received
		Latency	Min
			Max
			Average
	Stream	Total	Frames Sent
		Rate	Frames Sent
Capture	Frames		

Table 33: Ixia Commands

10.4.6.18 Burst Code Example

The following code example is a burst send and receive test.

```
01 /**
02 * Send burst of 1000 IP packets at a rate of 100 packets per se
cond, then the tests whether 1000 packets has been received.
03 *
04 * @throws Exception
05 */
06 public void testSimplePlay() throws Exception {
07 /*
08 * Configuration
09 */
10 report.step("Set transmit rate to 100 packets per second");
11 tg.port[0].ps[0].setRatePPS(100);
12 report.step("Set burst size to 100 packets");
13 report.step("Set Ethernet frame type");
14 tg.port[0].ps[0].frame.setL2Protocol(EnumL2.L2_PROT_ETH_II);
15 report.step("Set TCP frame type");
16 tg.port[0].ps[0].frame.setIPAddr("15.0.0.1", "13.0.0.1");
17 /*
18 * Execution
19 */
20 report.step("Set port 0 to send packets");
21 tg.aPlay.setPlayers(new int[] {0});
22 report.step("Set port 1 to receive packets");
23 tg.aPlay.setReceivers(new int[] {1});
24 report.step("Set modes and play");
25 tg.aPlay.setModes(false, false, false, false);
26 tg.aPlay.play();
27 /*
28 * Analysis
29 */
30 tg.aPlay.analyze(new TrafficCounter(1,
31 EthernetPortCounters.EnumType.RECIEVE_PACKETS, 1000, 0));
32 }
```

Table 34: Ixia Code Example

10.4.7 Ignis Iperf Driver

10.4.7.1 Iperf Overview

Iperf was developed by NLANR/DAST as a modern alternative for measuring maximum TCP and UDP bandwidth performance. Iperf allows the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss. Iperf is a basic traffic generator that operates via a command line interface.

Supports IPv6 Platform:

- **UNIX systems**
- **Windows**
- **Mac OS**

Iperf is an effective solution used for investigating circuit quality and when a new circuit has been established.

Mode	Measure Bandwidth
TCP Mode	Reports MSS (Maximum Segment Size)/MTU (Maximum Transfer Unit) size and observed read sizes
	Supports TCP window size via socket buffers
UDP Mode	Client can create UDP streams of specified bandwidth
	Measure packet loss, delay, jitter

Table 35: Iperf Modes

10.4.7.2 Ignis Iperf Driver Overview

The **Ignis Iperf Driver** provides an API with parallel functionality to the Iperf command line application. This enables the user to configure, run, read and analyze Iperf traffic results. The **Ignis Iperf Driver** is interfaced to the Iperf endpoints stations (client/server) using JSystem a CLI Connection enabling the driver can control and managed the Iperf server side and client side of configurations.

The traffic results saved on the server side station are saved in a text file format; at the end of the Iperf run the result file is transferred to the J-System JRunner station by the FTP in order to accommodate the analysis of the results.

10.4.7.3 Iperf Specification Table

The table shows the supported Iperf specifications.

Specification	Description
Supported Versions	1.7.0_TOS_Enabled 1.7.0
Operating Systems	Windows XP
	Linux
Pre-Requisite	Telnet agent on the machines on which Iperf application is running (Client machine and Server Machine). Setting the iperf.exe directory path in the System variables path.

Table 36: Iperf Application Specifications

10.4.7.4 Iperf System Object Operations

The table shows all of the supported Iperf entities.

Iperf Entity	
Iperf	init
	setConnectivityParams
	connect
	start
	connectAndStart
	getIperfResultFile
	isRunning
	reconnect

Table 37: Iperf Entities

10.4.7.5 List of Iperf Attributes Commands

The following table lists the Iperf commands that are supported by Ignis.

Attribute	Ignis Driver Support	Side	Description
-f	No	Client/Server	format to report: Kbits, Mbits, KBytes, MBytes
-i	YES	Client/Server	seconds between periodic bandwidth reports
-l	YES	Client/Server	length of buffer to read or write (default 8 KB)
-m	YES	Client/Server	print TCP maximum segment size (MTU - TCP/IP header)
-o	NO	Client/Server	output the report or error message to this specified file
-u	YES	Client/Server	use UDP rather than TCP
-w	YES	Client/Server	TCP window size (socket buffer size)
-B	YES	Client/Server	bind to <host>, an interface or multicast address
-C	YES	Client/Server	for use with older versions does not sent extra msgs
-M	NO	Client/Server	set TCP maximum segment size (MTU - 40 bytes)
-N	NO	Client/Server	set TCP no delay, disabling Nagle's Algorithm
-V	NO	Client/Server	Set the domain to IPv6
-s	YES	Server	run in server mode
-D	NO	Server	run the server as a daemon
-R	NO	Server	remove service in win32
-b	YES	Client	for UDP, bandwidth to send at in bits/sec
-c	YES	Client	run in client mode, connecting to <host>
-d	NO	Client	Do a bidirectional test simultaneously
-n	YES	Client	number of bytes to transmit (instead of -t)
-r	NO	Client	Do a bidirectional test individually
-t	YES	Client	time in seconds to transmit for (default 10 secs)
-F	YES	Client	input the data to be transmitted from a file
-L	NO	Client	input the data to be transmitted from stdin
-P	YES	Client	number of parallel client threads to run
-T	YES	Client	time-to-live, for multicast (default 1)
-S	YES	Client	Wmm -Quality of service

Table 38: Iperf Attribute Commands

10.4.7.6 Code Example

The following code example shows a standard run mode

```
01 package com.aqua.iperf.example;
02 import java.io.File;
03 import com.aqua.iperf.Iperf;
04 import com.aqua.iperf.IperfAnalyzer;
05 import com.aqua.iperf.IperfCounters;
06 import com.aqua.iperf.IperfFile;
07 import junit.framework.SystemTestCase;
08 /**
09  *
10  * @author HaimM
11  *
12  * The test demonstrate iperf regular running  mode .
13  * You can run this demo in the Runner in two different ways:
14  * 1) running all of the tests in one test  .
15  * 2) running each test as separate test in the runner .
16  *
17  */
18
19 public class IperfSingleStream extends SystemTestCase{
20 private static Iperf iperf = new Iperf() ;
21 /**
22 * Connecting the Jsystem Runner to the the server machine and c
lient machine
23 */
24 protected void connect() throws Exception {
25 report.report("connecting to server machine and client machine"
);
26 iperf.setConnectivityParams("172.16.15.80", "telnet", "METALINK
DSL\\lab", "metlab","windows",
27 "172.16.15.45", "telnet", "METALINKDSL\\lab", "metlab","windows
");
28 iperf.connect();
29 }
30 /**
31 * The iperf object save the resulrts file of the server side be
cause when runnig UDP
32 * just the server side knows the received throughput , packet l
oss and jitter delay .
33 *
34 *  This method bring the server side result file and determine
which kind
35 *  of analysis to do .   analisys type can be : IperfFile.AVERA
GE , IperfFile.BEST , IperfFile.WORST
36 */
37 public void testRun() throws Exception {
38 iperf.init();
39 iperf.setThrowException(false);
40 connect();
41 iperf.setTransmitMode("CONTINUOUS");  //CONTINUOUS OR BURST
42 iperf.setPending(true);
```

```
43 iperf.setWindowSize("128M");
44
45 report.report("starting to generate traffic ");
46 iperf.start( "10.0.2.101" ,"-
u", "45M"    ,  1            , 20     , 1400,"./iperf.txt"  );
47 /* for using TCP you have to leave the second field empty  *
48 iperf.start( "127.0.0.1" ," ", "95M"    ,  1            , 60
  , 1400      ,"C:\\aqua\\iperf.txt");
49 */
50
51 /* if you want to run all of the code in one test, expose the t
ests below */
52 testAnalyzeResultFile();
53 iperf.close();
54 }
55 public void testAnalyzeResultFile() throws Exception {
56 //                                                  what to an
alyze      ,  expectedResult,tollerance
57 iperf.analyze(new IperfAnalyzer( IperfFile.AVERAGE ,IperfCounte
rs.THROUGHPUT  , 45            , 0.5 ));
58 iperf.analyze(new IperfAnalyzer( IperfFile.BEST    ,IperfCounte
rs.LOSSES      , 0.1           , 0.2 ));
59 iperf.analyze(new IperfAnalyzer( IperfFile.BEST    ,IperfCounte
rs.JITTER      , 0.50           , 0.9 ));
60 }
61 public void testCloseIperf() throws Exception{
62 iperf.close();
63 }
64 }
```

Table 39: Iperf Code Example

10.4.8 Ignis Hammer Driver

10.4.8.1 Hammer Load Testers Overview

The Hammer Family offers an integrated package for pure feature and load testing, in addition to more sophisticated scenarios of functional-load testing. Built on Empirix's Hammer FX (VoIP feature test) and Hammer NXT (high density VoIP load generation) products, this new platform offers the combined capabilities of both in a unified user and programming interface.

Stress testing of VoIP and IMS products and services requires high call-volumes, realistic simulation of end-user devices, quick test setup, and comprehensive reporting at a price that can fit a lab budget. By combining carrier class volumes of IP traffic with detailed analysis, diagnostics, and reporting, the Hammer G5 test platform delivers industry leading performance and capability.

Hammer G5 is capable of producing a combination of VoIP signaling and real media on every call, scaling from thousands to more than 200,000 simultaneous calls.

10.4.8.2 Hammer Load Testing Solutions

The following table details the Hammer load testing solutions.

Hammer Model	Description
Hammer G5	Addresses the need for integrated, comprehensive, and flexible call profiles including feature and load on a single carrier-class testing platform that can scale from 4 to more than 200,000 endpoints.
Hammer FX-TDM	Addresses the need for simulating hundreds to thousands of simultaneous callers.
Hammer Registration Tester	Addresses the need for generating high volumes of registration traffic and background registration load
Hammer DEX	Addresses the need for emulating device-to-device interactions
Hammer Call Analyzer	Addresses the need for troubleshooting and diagnostics during tests
Hammer NetEm	Addresses the need for simulating real-world IP network impairments and problems

Table 40: Hammer Load Testing

10.4.8.3 Hammer Load Test

The following illustration shows the Hammer load test signal flow.

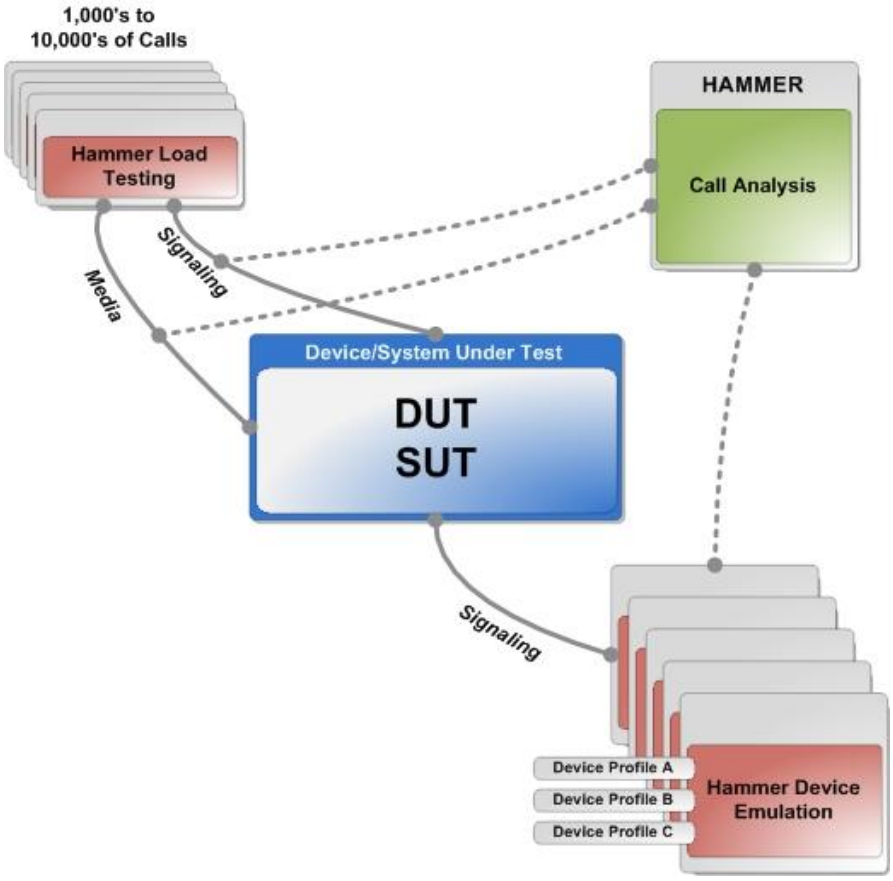

Figure 8: Hammer Signal Flow

10.4.8.4 Features and Specifications

The following table details the features and specifications of the Hammer driver.

Specification	Description
Signaling Protocols	Supports SIP UA, IMS SIP UE), MGCP, NCS, H.323, Cisco's SCCP (Skinny), QSIG, and Clear Channel (No Call Control).
	Transport over UDP and TCP, IPv4 and IPv6, and TLS for secure transport.
	Unique endpoint emulation using IP address, MAC address, and VLAN tagging.
	Mixed protocol configurations supported.
	Highly configurable message content and sequence.
Media Capability	G.711 A/μ-law, G.723.1, G.726, G.729A, GSMAMR/ EFR/FR, EVRC0, RFC 2833, and user-defined.
	codecs for voice and tones.
	H.263 and MPEG4 pt2 codecs for video.
	T.38 Fax.
	Silence suppression detection using RFC 3389, G.723.1A, and G.729B.
	Multiple true voice and video clips.
	Unique DTMF string validation on every channel.
	Reference-based voice quality using ITU-T P.862 (PESQ).
	Non-reference-based voice quality using E-model (R-factor).
	Advanced speech recognition.
	Full RTP and RTCP metrics for every call.
	Narrowband and Wideband codecs for media.

Scalability	Delivered in 4 to 24,000 endpoints for a single server.
	Scales to >200,000 endpoints with use of Master/Mega-Controller for single point of control.
	Manage >20 Hammer systems with single point of control from Mega-Controller.
Hammer G5 User Interface	Easy to use TestBuilder UI allows for test scripts to be created by a ladder diagram - all scripts are protocol independent.
	Low-level complex scripting can be developed and executed using the Hammer Visual Basic (HVB) UI.
	Multiple test scripts can be run from the TestBuilder or Test Profiler UI.
	Real-time test information is displayed in the System Monitor window for each channel, in the Call Summary Monitor for summarized call metrics, and in the QoE Monitor window for aggregated results over time.
	Configuration, Test script, and Profile files can be saved and opened on a different system.
	Powerful Command Line Interface (CLI) and scripting interface for automation, remote control, and feature testing.
Real-Time Monitoring and Reporting	Real-time monitoring and reporting of statistics.
	Statistics can be viewed on any pair of endpoints.
	Export data to other applications for customized user report.
Signaling Editor	Customize State Machine Behavior and Message Content
	Flexible state based architecture provides comprehensive state machine customization capability
	Develop state machines to handle call flows specific to applications and device under test
	Create individual message or call flows, or customize existing ones
	Import messages from Hammer Call AnalyzerTM (HCATM) or third-party protocol analyzers

Table 41: Hammer Specification and Features

10.4.9 Ignis N2X Driver

10.4.9.1 N2X Overview

The Agilent N2X provides the ultimate solution for validating the performance and scalability characteristics of next-generation network equipment for voice, video and data (triple-play) services. N2X addresses the complex challenges of validating next generation equipment by providing a single test environment to simultaneously validate leading edge services over the latest infrastructures.

Network equipment manufacturers and service providers can gain unique insight into quality of experience (QoE) of each individual subscriber service under real-world conditions. N2X addresses the test challenges triple-play services impose across the IP/MPLS core, carrier edge and broadband-access networks, enabling a more complete characterization of service quality and the networking mechanisms required to deliver it. With the industry's most powerful integrated data and control plane test capability, N2X uniquely validates QoS mechanisms and high availability implementations. Using powerful emulation software, purpose built applications and industry-leading hardware test cards, users can test a broader range of test cases in much less time than ever before.

- **Unique Realism** - Simultaneously test a wide variety of services across broadband access, carrier edge and IP/MPLS core.
- **Superior Scalability** - Comprehensive protocol coverage to emulate the ultimate scale and complexity of network services.
- **Rapid Time to Insight** - Powerful measurement and analysis tools to rapidly isolate problems anywhere in the network.
- **Communication** - Support for SONET/SDH, POS, ATM, Ethernet, Frame Relay and Fiber Channel interfaces.

10.4.10 Ignis TestCenter Driver

10.4.10.1 TestCenter Overview

Spirent TestCenter is a network performance analysis system that includes software and hardware components to simplify and accelerate the testing process. With Spirent TestCenter, functional and performance testing are centralized through the use of a single application that can generate control and data plane traffic and collect test results.

Spirent TestCenter can be used to create large-scale tests that exceed the capacity of devices under test. Such tests can evaluate key performance parameters under typical and extreme subscriber traffic load conditions, and they can determine whether connection rates can support expanding subscriber bases at various bandwidth levels.

The Spirent TestCenter system supports testing of the extended technologies required of today's routers, including QoS, IPv4/IPv6 routing, multicast, MPLS, spanning tree, VLAN, VLAN stacking and DHCP. Spirent TestCenter also effectively tests Ethernet switches that are capable of up to 1-Gigabit Ethernet connections to the desktop and 10-Gigabit Ethernet connections to the core.

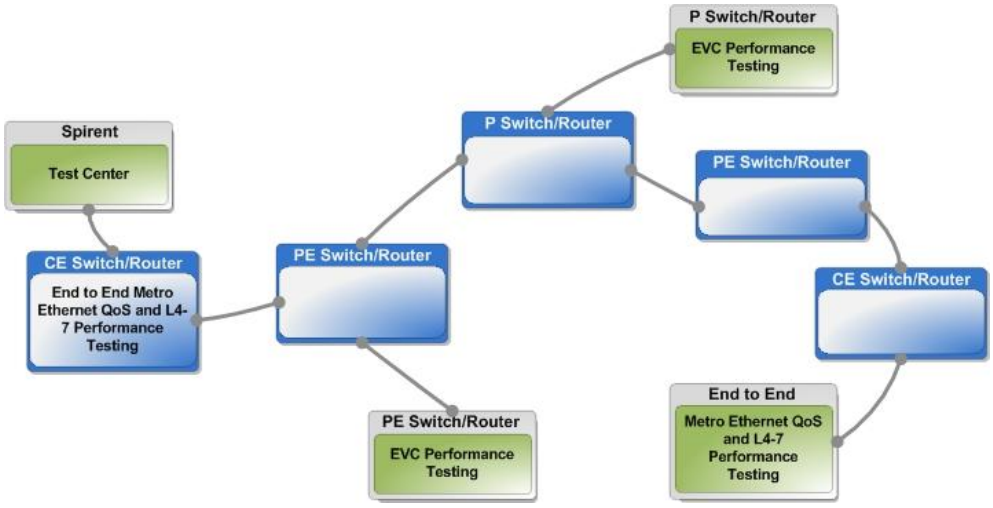

Figure 9: Spirent TestCenter Ethernet Testing

The number of multimedia service subscribers is growing, service providers must be able to scale their networks and deliver services reliably, regardless of the aggregate bandwidth requirements. Building reliability into Ethernet-based broadband networks requires appropriate testing in the lab and in the field to ensure the seamless delivery of new and existing rich IP services. The TestCenter network provides a flexible and expandable framework for growing tests as required.

10.4.10.2 Key Features

- Supports up to 144 Dual Media GigE ports or up to 24 10GbE ports in a single chassis
- Each port can generate up to 32,767 TCP/IP streams or analyze up to 65,535 streams in real time
- Supports high-density 10/100/1000, GigE and 10GigE test modules
- Supports hot-swappable modules
- Wire rate traffic generation and analysis
- Full IPv4/IPv6 dual stack support
- GUI provides highly granular traffic generation controls, and real-time and historical analysis tools
- Powerful wizards to speed tasks such as system configuration and which allow easy setup of unicast, multicast, IPv4 and IPv6 stream
- Spirent TestCenter Automation uses a Tcl interface that supports precise control over test generation and execution
- Reduce test cycles, easily test RFC compliance, and create regression tests without programming

10.4.10.3 Ignis Cisco Catalyst 3500 Driver

10.4.10.4 Cisco Catalyst 3500 Driver Overview

The Cisco Catalyst 3500 Series XL is a scalable line of stackable 10/100 and Gigabit Ethernet switches that deliver premium performance, manageability, and flexibility, with excellent investment protection. Catalyst 3500 XL stackable switches allow management of all switched ports from a single IP address and provide interconnected switches with an independent high-speed stack bus that preserves valuable desktop ports.

Cisco Catalyst 3500 Series XL switches utilizes Cisco Switch Clustering technology and GigaStack GBICs. With Cisco Switch Clustering, users can manage over 380 ports from a single IP address, and connect up to 16 switches, regardless of physical location, with a broad range of Ethernet, Fast Ethernet, and Gigabit Ethernet media. Cisco Switch Clustering is supported on all Catalyst 3550, 2950, 3500 XL, 2900 XL and 2900 LRE XL switches.

The Catalyst 3508G XL is a single RU, stackable Gigabit Ethernet switch with eight GBIC-based Gigabit Ethernet ports. The Catalyst 3508G XL is ideal for aggregating a group of 10/100 and Gigabit Ethernet switches and Gigabit Ethernet servers through Cisco GigaStack™ GBICs or standard 1000Base-X GBICs.

10.4.10.5 Feature and Specifications

The table details the features and specifications of the Catalyst 3500 driver.

Specification	Description
Performance	10-Gbps shared memory switching fabric with 5-Gbps peak forwarding bandwidth
	7.5-Mpps wire-speed forwarding rate for 64-byte packets
	4 MB memory architecture shared by all port
	Packet forwarding rate for 64-byte packets: 1,488,000 pps to 1000Base-X ports
	8 MB DRAM and 4-MB Flash memory
	8192 MAC addresses
Management	SNMP Management Information Base (MIB) II, SNMP MIB extensions, bridging MIB (RFC 1493)
Standards	IEE 802.3x full duplex
	IEEE 802.1D Spanning-Tree Protocol
	IEEE 802.1p Class of Service Protocol
	IEEE 802.1Q VLAN
	IEEE 802.3z, IEEE 802.3x 1000Base-X specification
	1000Base-X (GBIC)
	1000Base-T
	1000Base-SX
	1000Base-LX/LH
	1000Base-ZX (4 GBICs supported per switch)
	IEEE 802.3ab

Table 42: Cisco Catalysis 3500 Specifications

10.4.11 Ignis Adtech AX 4000 Driver

10.4.11.1 Adtech AX4000 Overview

The AX/4000 is completely modular and designed for maximum flexibility and scalability. It is a multi-port system that can currently test four different transmission technologies (IP, ATM, Ethernet, and Frame Relay) simultaneously at speeds up to 2.488 Gbps. The AX/4000 also supports multiple users and multiple control options including Windows,® UNIX, and any C or Tcl compatible platform. The user can even create a virtual AX/4000 system spanning the globe by linking multiple AX/4000 systems in different cities and operating them as a single cohesive unit.

10.4.11.2 Adtech AX4000 Specification

- Broadband Access
- Core Network
- IPv6 Testing
- Metro Area Network
- Optical
- Routing (Performance & Conformance)
- 3G Wireless
- Content
- Performance Analyzer & Protocol Analyzer
- AX/4000 Broadband Test System Products
- AX/4000 Broadband Test System
- Reference: Adtech AX4000 (AX/4000) System (pdf)

10.4.11.3 Code Example

The following code is an example of a basic activation for the Adtech AX4000.

```
01 public class AdtechBasicTests extends SystemTestCase{
02     Adtech adtech;
03     public void setUp() throws Exception{
04         adtech = (Adtech)system.getSystemObject("adtech");
05     }
06
07     public void testBasicGeneration() throws Exception{
08         adtech.ports[0].interfaceCreate(InterfaceType.IPoETHER)
;
09         adtech.ports[1].interfaceCreate(InterfaceType.IPoATM);
10
11         adtech.ports[0].interfaceReset();
12         adtech.ports[1].interfaceReset();
13
14         adtech.ports[0].interfaceSetGiga(InterfaceMode.NORMAL,
true);
15         adtech.ports[1].interfaceSetOc12NonChannelized(
16                 ClockSource.INTERNAL,
17                 InterfaceMode.NORMAL,
18                 AtmNonChannelizedInterfaceOpmode.SONET,
19                 AtmChannelizedInterfaceType.SONETOC12, true, fa
lse, false, false, 19, 5, 5, 0, 0, 19, 3);
20         adtech.ports[0].genCreate();
21         adtech.ports[1].genCreate();
22
23         adtech.ports[0].genDefineIpoetherTestPkt(
24                 1, EthHeaderType.EII, -1, false,
25                 ValueOptions.FIXED, "00:00:00:00:00:AA", null,
null, -1, null,
26                 ValueOptions.FIXED, "00:00:00:00:00:01", null,
null, -1, null,
27                 true, 16, 0, 0, 33024);
28         adtech.ports[0].genDefineIpBlk(1,
29                 ValueOptions.FIXED, "0.0.0.1", null, null, -
1, null,
30                 ValueOptions.FIXED, "0.0.0.0", null, null, -
1, null,
31                 -1, -1, 0, -1, 64, -1, 173, 0, 0, -1);
32         adtech.ports[0].genDefineIpTstBlk(1, FillOptions.RAND,
-1);
33         adtech.ports[0].genDefineDatagramLength(1, 250, 3.37);
34         adtech.ports[0].genDist(1, 20666);
35         adtech.ports[0].genSeqSend(1);
36         adtech.ports[1].genDefineIpoatmTestPkt(1, 20, 40);
37         adtech.ports[1].genDefineLlcSnapBlk(1, 0xAAAA03, 0x0080
C2, 0x0007);
38         adtech.ports[1].genDefineUserBlk(1, "{0000}", null);
39         adtech.ports[1].genDefineIpEthBlk(
40                 1, EthHeaderType.EII, -1,
```

```
41                 ValueOptions.FIXED, "00:22:33:44:55:66", null,
null, -1, null,
42                 ValueOptions.FIXED, "00:77:88:99:00:11", null,
null, -1, null,
43                 true, 100, 0, 0, 33024);
44         adtech.ports[1].genDefineIpBlk(1,
45                 ValueOptions.FIXED, "1.1.1.2", null, null, -
1, null,
46                 ValueOptions.FIXED, "1.1.1.1", null, null, -
1, null,
47                 -1, -1, 0, -1, 64, -1, 173, 0, 0, -1);
48         adtech.ports[1].genDefineIpTstBlk(1, FillOptions.RAND,
-1);
49
50         adtech.ports[0].anaCreate();
51         adtech.ports[1].anaCreate();
52         adtech.ports[0].anaRun();
53         adtech.ports[1].anaRun();
54         sleep(5000);
55         adtech.ports[0].genRun();
56         adtech.ports[1].genRun();
57         sleep(10000);
58
59         adtech.ports[0].anaStats();
60         adtech.ports[0].analyze(new CheckCounter("totalPackets"
, 0));
61         adtech.ports[1].anaStats();
62         adtech.ports[1].analyze(new CheckCounter("totalPackets"
, 0));
63     }
64
```

Table 43: Adtech Code Example

10.4.12 Ignis Ant 20 Driver

10.4.12.1 Ant 20 Overview

Since its launch, the ANT-20 Advanced Network Tester has established itself as the leading tester for advanced digital networks. Packed with power and features, the ANT-20 is a powerful all-rounder for SDH and SONET backbone testing.

Application areas include development labs, acceptance, conformance, and functional testing in production, and even the troubleshooting of in-service networks. The ANT-20 product family delivers today what other products promise for tomorrow, including SDH, SONET, PDH, ATM, and Jitter.

- Multi-rate transmission testing from DS1 to OC-192c and E1 to STM-64c.
- Modular platform offering SONET, DSn, SDH, and ATM capabilities.
- Large color - touch screen display.
- Intuitive, easy to use graphical user interface (GUI).
- Flexible integration into lab or manufacturing test environments.
- Modular hardware and software concept.
- Portable mainframe.

10.4.12.2 Key Features

The Ant 20 has a built-in Pentium PC and Windows-based GUI for easy processing of test results, and a graphical results presentation that enables offline analysis on an external PC and allows for the post-processing of results.

- Access to all SOH and TOH bytes.
- Generation and analysis of errors and alarms.
- Performance measurement according to G.821, G.826, G.828, G.829, M.2100, and M.2101.
- Pointer sequencing according to G.783 and graphical pointer analysis.
- BERT and V.11 interface for DCC.
- Ethernet, GPIB, and RS-232/V.24 interfaces allow for easy adaptation to test automation environments.
- Easily adapted to cover a new test scenario, new standards, higher bit rates, and intelligent systems components.

10.4.12.3 Applications

- **BERT and Performance Testing** - Test for DSn, PDH, SONET, and SDH signals support of bit error rate for all standard bit rates along with performance tests according to the latest ITU-T recommendations.
- **Jitter and Wander Analysis** - Generation and analysis of jitter and wander for bit rates from 1.5 Mbps to 10 Gbps fully compliant to ITU-T Recommendation 0.172.
- **Analysis of Tandem Connection Monitoring (TCM) -** Verifies the proper implementation of TCM in the SDH network.
- **Comprehensive ATM Testing** - Tests ATM networks for correct operation and Quality of Service (QoS). Depending on the application, the ANT-20 has test solutions for permanent virtual circuits (PVC) and switched virtual circuits (SVC).
- **Test Automation** - Time and cost savings through automation. The CATS test sequencer is test automation software designed to run on the ANT-20's built-in PC. It is the ideal tool for automating repeated test procedures.
- **Remote Operation** - Due to the Microsoft Windows-based design, the same software can be run on both the ANT-20 and on a laptop or PC, enabling complicated and time-consuming tests to be carried out remotely.

10.4.12.4 Ignis Ant 20 Driver Overview

The **Ignis Ant 20 Driver** is a TDM traffic simulator. The functionality that it supports is divided into two main categories, OC3 (Optical Carrier) and PDH (Plesiochronous Digital Hierarchy).

- **OC3 Supports** – Sonnet and Synchronous Digital Hierarchy (SDH).
- **PDH Supports** – E-1 and T-1.

10.4.12.4.1 PDH Network System

The Plesiochronous Digital Hierarchy (PDH) is a technology used in telecommunications network to transport large quantities of data over digital transport equipment such as fiber optic and microwave radio systems.

Note: PDH allows transmission of data streams that are nominally running at the same rate, but allowing some variation on the speed around a nominal rate.

The **Ignis Ant 20 Driver** supports all streaming configurations, line coding and both balanced and unbalanced port types.

The following table shows the code formats types that are supported by the **Ignis Ant 20 Driver** PDH networking functionality:

Code Type	Description	
T-1	B8ZS AMI	Unframed
		SF
		ESF
E-1	HDB3 AMI	Unframed
		PCM30
		PCM30-CRC
		PCM31
		PCM31-CRC

Figure 10: Ignis Ant 20 PDH Driver Configurations

10.4.12.4.2 PDH Functionality Support

The PDH functionality supported by the **Ignis Ant 20 Driver** is as follows:

- Supports all frame traffic types.
- Supports line sequences PRB15, PRB20 and QRSS20 (DS-1 only).

Supported Functionality	
PDH (Plesiochronous Digital Hierarchy)	Stream Configuration
	Alarm Generation
	Error Generation
	Alarm Status Retrieval
	Error Status Retrieval
	Full System Analysis

Table 44: Ant 20 PDH Supported Functionality

10.4.12.4.3 OC3 Network System

Optical Carrier (OC) levels describe a range of digital signals that can be carried on SONET fiber optic network. The number in the Optical Carrier level is directly proportional to the data rate of the bitstream carried by the digital signal.

The following table shows the code formats types that are supported by the **Ignis Ant 20 Driver** OC3 networking functionality:

Code Type	Description
SONET (84 T-1 Streams)	Stream Configuration VT-1
	Stream Configuration STS-3C
SDH (63 E-1 Streams)	E-1 Streams

Table 45: Ignis Ant 20 OC3 Driver Configurations

Supported Functionality	
Optical Carrier (OC)	Stream Configuration
	Alarm Generation
	Error Generation
	Alarm Status Retrieval
	Error Status Retrieval
	Full System Analysis
	All Stream Format Configurations
	Active Channel Configurations
	Sequenced Testing via Streamed Traffic
	OC3 Support for Full Data Integrity Verification for each Stream.

Table 46: Ant 20 OC3 Supported Functionality

10.4.12.5 Code Example

The following code example shows a standard Ant system object implementation. .

```
01 package com.aqua.sysobj.ant;
02 import junit.framework.SystemTestCase;
03 import com.aqua.sysobj.ant.intrfc.InterfaceConfig;
04 public class AntBasicTests extends SystemTestCase {
05
06   Ant ant;
07   public void setUp() throws Exception{
08     ant = (Ant)system.getSystemObject("ant");
09   }
10
11   public void testScipCommand() throws Exception{
12     ant.intrfc.setClockSource(InterfaceConfig.CLK_INTERNAL);
13     String clk =  ant.intrfc.getClockSource();
14     if (clk.compareTo(InterfaceConfig.CLK_INTERNAL) == 0){
15       report.report(ant.basic.getSystemTime() +": Config OK");
16     }
17     else
18       report.report("Fail to config");
19   }
20   public void testCommandError() throws Exception{
21     //scpi.handleScpiCommand("sd","ccc :SOUR:CLOCK:SOUR RX?", 1
0000);
22   }
23
24   public void testSystemDateAndTime() throws Exception {
25     report.report("Date/Time: "+ ant.basic.getSystemDate() + an
t.basic.getSystemTime());
26   }
27
28   public void testAllHistoryAlarmsFail() throws Exception {
29
30     ant.basic.checkAllHistoryAlarms(20, true);
31   }
32
33   public void testAllHistoryAlarmsNotFail() throws Exception {
34
35     ant.basic.checkAllHistoryAlarms(20, false);
36   }
37
38   public void testAllCurrentAlarmsNotFail() throws Exception {
39
40     ant.basic.checkAllCurrnetAlarms(false);
41   }
42
43 }
```

Table 47: Ant 20 Code Example

10.5 Web Drivers

The web drivers are part of the application layer that enables the user the ability to connect to the web and utilize web technologies. The illustration shows the relevant application web drivers in their relative position to the layers of the JSystem Automation Platform.

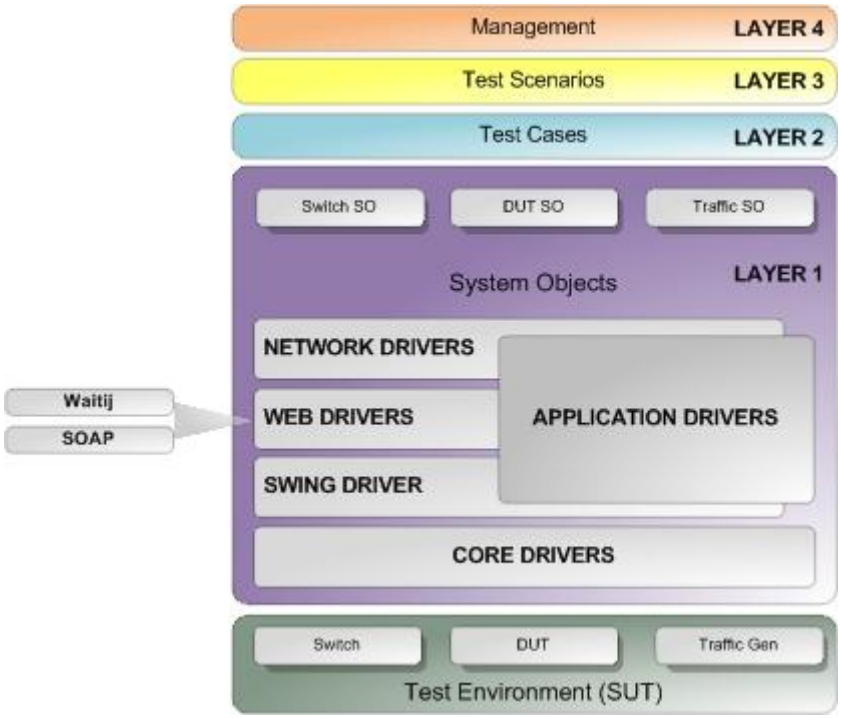

Figure 11: Web Application Drivers Layer

10.5.1 Ignis Watij Web Driver

10.5.1.1 Watij Overview

Watij (pronounced wattage) stands for Web Application Testing in Java. Watij is a pure Java API created to allow for the automation of web applications. Based on the simplicity of Watir and enhanced by the power of Java, Watij automates functional testing of web applications through a real browser. Currently Watij supports automating Internet Explorer on Windows only.

10.5.1.2 Selenium Open GA Overview

Selenium is a test tool for web applications. Selenium tests run directly in a browser, just as real users do, such as Internet Explorer, Mozilla and Firefox on Windows, Linux, and Macintosh. No other test tool covers such a wide array of platforms.

- **Browser Compatibility Testing –** The user has the ability to test an application to see if it works correctly on different browsers and operating systems. The same script can run on any Selenium platform.
- **System Functional Testing –** The user can create regression tests to verify application functionality and user acceptance.

10.5.1.3 Supported Platforms:

The following table details the supported operating systems and their related browsers that are supported by JSystem.

Platform	Browser Version
Windows	Internet Explorer 6.0, 7.0
	Firefox 0.8 to 2.0
	Mozilla Suite 1.6+, 1.7+
	Seamonkey 1.0
	Opera 81
Mac OS X	Safari 1.3+
	Firefox 0.8 to 2.0
	Camino 1.0a1
	Mozilla Suite 1.6+, 1.7+
	Seamonkey 1.0
Linux	Firefox 0.8 to 2.0
	Mozilla Suite 1.6+, 1.7+
	Konqueror

Table 48: Ignis Web Driver Supported Browsers

10.5.1.4 Ignis Watij Driver Overview

Ignis Watij Driver uses several open source project that support Web application testing. The services supplied by these projects are abstracted by the **Ignis Watij Driver**.

The open source projects that are used to support the Ignis JSystem Framework are:

- **Watij** - http://www.watij.com/
- **Selenium** - http://www.selenium.openqa.org/

10.5.1.5 Code Example

The following code example illustrates how to connect to a site and click on a link name, and take a screen shot.

```
01 package com.aqua.webapp.tests;
02 import junit.framework.SystemTestCase;
03 import com.aqua.webapp.analyzers.FindTextInBrowser;
04 import com.aqua.webapp.systemobject.WebAppSystemObject;
05 public class GoogleTest extends SystemTestCase {
06 WebAppSystemObject webApp1;
07 WebAppSystemObject webApp2;
08 protected void setUp() throws Exception {
09 super.setUp();
10 webApp1 = (WebAppSystemObject) system.getSystemObject("webapp1"
);
11 webApp2 = (WebAppSystemObject) system.getSystemObject("webapp2"
);
12 }
13 /**
14 * A simple example, shows how to connect site, and click on Lin
k name with
15 * the possibiluty for screen capture.
16 *
17 * @throws Exception
18 */
19 public void testGoogle() throws Exception {
20 webApp1.connect();
21 webApp1.clickOnLinkName("Images", true);
22 webApp1.clickOnLinkName("Groups");
23 webApp1.clickOnLinkName("News", true);
24 }
25
26 /**
27 * Example for analyzing text on  browser`s text.
28 * @throws Exception
29 */
30 public void testClickOnLinkUsingXpath() throws Exception {
31 webApp2.connect();
32 webApp2.clickOnLink("//A[@href=\"aboutus.php\"]");
33 webApp2.analyze(new FindTextInBrowser("About AQUA"));
34 }
35 public void testRfresh() throws Exception{
36 webApp2.connect("http://yhjkh.fdf.df");
37 webApp2.clickOnLink("A");
38
39 }
```

Table 49: Ignis Web Driver Code Example

10.5.2 Ignis XML RPC/ SOAP Driver

10.5.2.1 XML RPC/Soap Overview

XML RPC and SOAP allow the user to perform remote procedure calls as XML messages are sent as part of a regular HTTP POST. The HTTP refers to XMLRPC and SOAP as HTTP POST requests that have XML data in them rather than the results of a HTML form.

XMLRPC is a simple version of SOAP. Unlike XMLRPC SOAP allows you to use other transport mechanisms than HTTP (SMTP, FTP, Jabber). SOAP also is not tied directly to the client/server model, so messages can be passed in other configurations such as a peer-to-peer network. To achieve these ends SOAP uses more advanced features of XML than XMLRPC (namespaces, schemas).

SOAP can be just as simple as XMLRPC, but it can also be more complicated if the user's needs requires it.

10.5.2.2 Ignis XML-RPC/SOAP Driver Overview

The **Ignis XML-RPC/SOAP Driver** utilizes the built-in java capabilities from several open source packages to interface with applications that expose the SOAP API.

The open source package used is the Apache's XML-RPC project. For more information about the Apache XML-RPC implementation follow the link to the Apache website: http://ws.apache.org/xmlrpc/

10.5.2.3 Code example

The following code example demonstrates the usage of a SOAP function.

```
01 public String echoMessage(String messageText) throws Exc
eption {
02 SOAPMessage message = applicationUtils.buildOperationMessage(Aq
uaCourseApplicationOperations.ECHO,messageText);
03 SOAPMessage response = sendMessage(message);
04 return applicationUtils.getResults(response);
05 }
06 private SOAPMessage sendMessage(SOAPMessage message) throws Exc
eption {
05 SOAPConnection connection =
06 soapConnectionFactory.createConnection();
07 java.net.URL endpoint = new URL(getEndPointUrl());
08 SOAPMessage response = connection.call(message, endpoint);
09 connection.close();
10 return response;
11 }
```

Table 50: XML RPC SOAP Code Example

10.6 Swing Driver

The swing driver refers to the Jemmy driver, this driver provides the ability to inter connect to the swing based applications GUI. This enables extended functionality and usability for testing. The illustration shows the relevant application swing driver in its relative position to the layers of the JSystem Automation Platform.

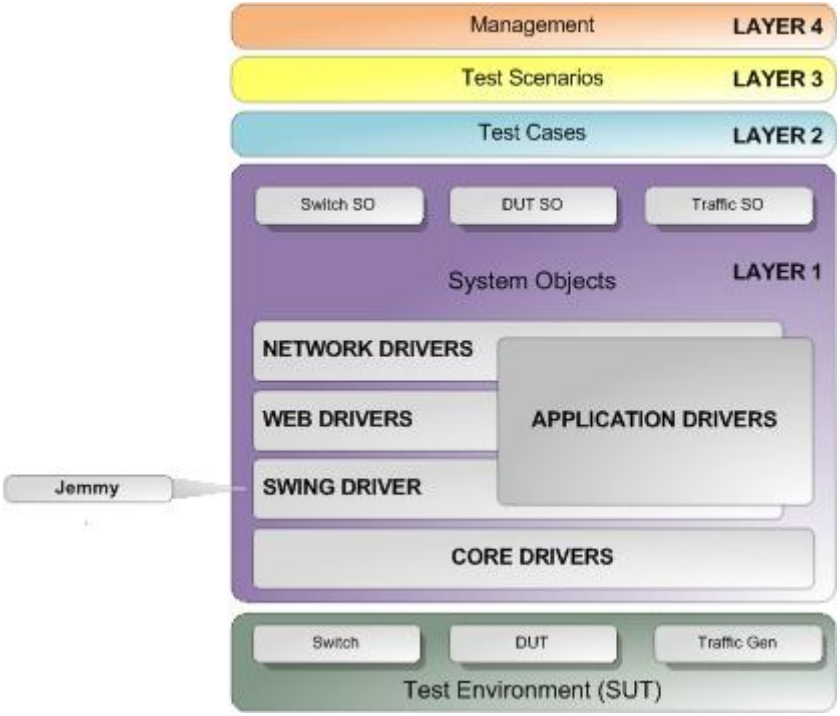

Figure 12: Swing Application Drivers Layer

10.6.1 Ignis Jemmy Driver

10.6.1.1 Jemmy Overview

Jemmy represents the most natural way to test Java UI - perform the testing right from the Java code. Jemmy is a Java library which provides clear and straightforward API to access Java UI. Tests are then just java programs, which use the API. Having the tests in Java allows to use all the flexibility of high level language to capture test logic and also do any other operations needed to be done from test.

10.6.1.2 Swing and AWT Support

Jemmy itself provides the entire API necessary to write tests in terms of Java UI components (Swing and AWT). All the Java UI components are covered to the extent it makes sense for UI testing.

10.6.1.3 Stability

Jemmy itself performs a lot of synchronizations with event thread (such as making sure that UI events reached component) and component's state (such as that typed text has actually accepted by text component) before proceeding with next test action. On top of that some public API is available to test developer to ensure more synchronization, if necessary.

10.6.1.4 Ignis Swing UI Driver Overview

At Ignis we believe that the majority of the automation should test the API level of the system under test, nevertheless covering parts of the user interface is part of almost all automation projects.

Ignis’s UI driver supports controlling UI application written with Java Swing.

In a typical setup the tests will run in one JVM and the AUT will run in a different JVM.

Ignis’s UI driver is composed of an agent that resides in the heap space of the AUT and a client which runs in the test JVM and sends commands to the agent. The agent in its turn emulates user operations on the application.

The client-server communication between the test and the SUT is implemented using the XML/RPC open source project, www.xmlrpc.com. Activating the swing application is done using the Jemmy open source project (link here).

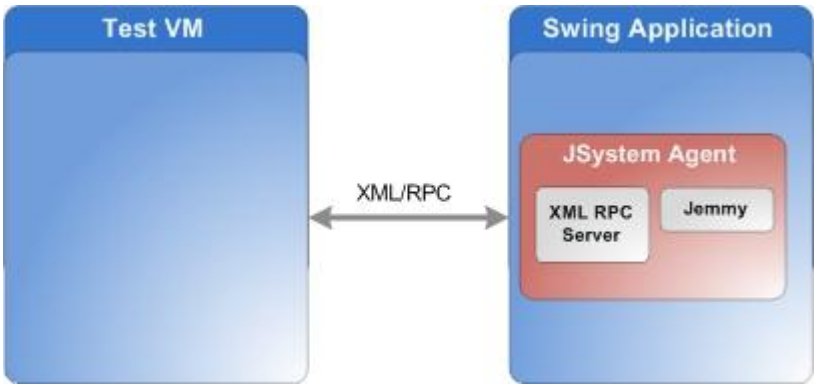

Figure 13: Jemmy Swing Driver

10.6.1.5 Client Code Example

The client code example shows a test that uses the Swing GUI driver. The test tests how the JRunner handles adding test which can't be instantiated.

```
01 /**
02    *
03    */
04   public void testAddErroredTest() throws Exception{
05 1.  jsystem.launch();
06 2.  report.step("clean scenario and add test which the system ca
n't load");
07 3.  jsystem.cleanCurrentScenario();
08 4.  jsystem.addTest("testShouldPass", "GenericBasic");
09 5.  jsystem.addTest("testDummyTest", "TestWhichFailsToInstantiat
e");
10 6.  jsystem.waitForWarningDialog();
11 7.  jsystem.play();
12 8.  jsystem.waitForRunEnd();
13 9.  jsystem.checkNumberOfTestExecuted(1);
14 }
```

Table 51: Jemmy Swing Driver Client Code Example

Line 1: The JRunner is launched.

Line 3: Clears the current scenario from all tests.

Line 4: Adds a normal test.

Line 5: Adds a faulty test.

Line 6: Waits for an error notification.

Line 7: Plays the scenario.

Line 8: Waits for the run to end.

Line 9: Verifies that only one test has been executed.

10.6.1.6 Server Code Example

In this code example we see the server side code of the "**waitForWarning**" API, when using Ignis's UI driver all the communication between the client and the server is transparent to the code writer, all the code writer has to take care of is activating Swing components using the Jemmy package.

```
01
02   /**
03    *  Waits for the runner's warning dialog to open and presses the OK b
utton.
04    */
05   public int waitForWarningDialog() throws Exception {
06     JDialogOperator dialog = new JDialogOperator(jmap.getWarningDialogWi
n());
07     new JButtonOperator(dialog, new
    TipNameButtonFinder(jmap.getDialogSelectOKButton())).push();
08     return 0;
09        }
```

Figure 14: Jemmy Swing Driver Server Code Example

10.6.2 Ignis Connectivity to Networking Test Equipment

One of the strengths of the **Ignis JSystem Automation Platform** is its proven ability to connect to any device using any type of interface, examples of the devices and interfaces are as follows:

- API: Tcl, Java, C, C++, Perl, .Net.
- CLI (over Telnet/ SSH/ RS232)
- SNMP (v1, v2, v3)
- HTTP
- OSS types: XML-RPC/ SOAP/ Corba
- RMI
- GPIB
- COMM
- WinSock: Client/Server, UDP/TCP
- DLLs

10.6.3 JSystem Stand Alone Functionality

Unlike other, generic, test automation platforms, the Ignis JSystem targets the testing of networking equipment, including tens of built-in drivers to control the standard test equipment used in testing networking equipment:

Element	Driver or Application Version	
Traffic Generators (For Testing Layers 1-3)	Ixia	Smartbits
	Adtech	Agilent N2X
	Ant	Hammer
Load Tools (Leading Load Tools for Layers 4-7)	IxLoad	
	Avalanche	
Clients (Standard Clients)	e-mail	Web
	Telnet	FTP
	DB	All end user client applications
WAN Emulators	Shunra Storm	PacketStorm
Sniffers and other Analysis Tools	Ethreal	TcpDump

Table 52: Ignis JSystem Stand Alone Functionality