Log4j

(expleo)

Contents

What is Logging?

[expleo]

What is Logging?

- Logging is a process that takes applications to a newer level with information logs on how the applications may or may not have performed/executed. It gives an exact idea of software performance, including any shortcomings.

- Log4j in Selenium is one such logging framework that helps in gathering information in the form of logs or log files.

What is Log4j in Selenium?

(expleo)

What is Continuous Integration

- Log4j is a logging framework written in Java that provides an easy way for logging in Selenium.
- In a nutshell, the framework gives out information about everything that goes on during the software execution.
- Log4j also provides insight into anything that may have gone wrong during software execution or automation.
- Overall, Log4j documents the output in the form of logs that can be examined later for purposes such as auditing small and large-scale Selenium projects.

Components of Log4j

[expleo]

CI Tools Comparison

The Log4j logging framework comprises the following components:

1. Logger
2. Appenders
3. Layout

1. Logger

The function of the logger in Log4j is basically storing and capturing all the necessary logging information that will be generated using the framework.

- **Logger Class** – To fully use the logger, create an instance for a logger class where all the generic methods will be at the user's disposal, required to use Log4j.

- **Log Levels** – These are the methods that will be used to print the log messages. There are primarily only a few log levels that are used in a script.

Logger levels -Log4j

1. **ALL –** This level will prioritize and include everything in the logs.

2. **ERROR –** This level will show messages that inform users about error events that may not stop the application.

3. **WARN –** This level will show information regarding warnings, that may not stop the execution but may still cause problems.

4. **DEBUG –** This level will log debugging information.

5. **INFO –** This level will log the progress of the application.

6. **FATAL –** This will print information critical to the system that may even crash the application.

Logger levels -Log4j2

OFF
FATAL
ERROR
WARN
INFO
DEBUG
TRACE
ALL

Syntax

To start logging messages using this basic configuration, all you need to do is obtain a Logger instance using the LogManager class:

```
private static Logger logger = LogManager.getLogger(MyService.class);
```

Then you can use the logger object with methods corresponding to the log level you want:

```
logger.error("This is an error message");
```


(expleo)

Appenders

The appender basically grabs information from the logger and writes log messages to a file or any other storage location.

The following are some of the appenders one can use for Log4j:

1. **FileAppender –** This will append the log messages to a file.

2. **RollingFileAppender –** It will perform the same function as FileAppender, but users will be able to specify the maximum file size. Once the limit is exceeded, the appender will create another file to write the messages.

3. **DailyRollingFileAppender –** It specifies the frequency by which the messages are to be written to the file.

4. **ConsoleAppender –** In this, the appender will simply write the log messages in the console.

Layout

The layout is where the format in which log messages will appear is decided.

There are several layouts one can use for log messages:

1. **Pattern Layout –** The user must specify a conversion pattern based on which the logs will be displayed. Otherwise, it takes the default conversion pattern in case of “no pattern specified”.
2. **HTML Layout –** In this layout, the format of logs will be in the form of an HTML table.
3. **XML Layout –** This will show the logs in an XML format

How to Set up Log4j in Selenium

(expleo)

Dependency to be added in pom.xml -log4j

```
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.19.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.19.0</version>
</dependency>
```

How to Use Log4j in Selenium

(expleo)

Follow the steps below to successfully run Log4j with Selenium Webdriver:

1. **Write an automation script**

2. After creating the script, **create a log4j.properties file** and **specify the root logger, appended, and layout information in the file**.

3. Import log4j dependencies like **Logger, PropertyConfigurator, and add them to the script along with the logger class**.

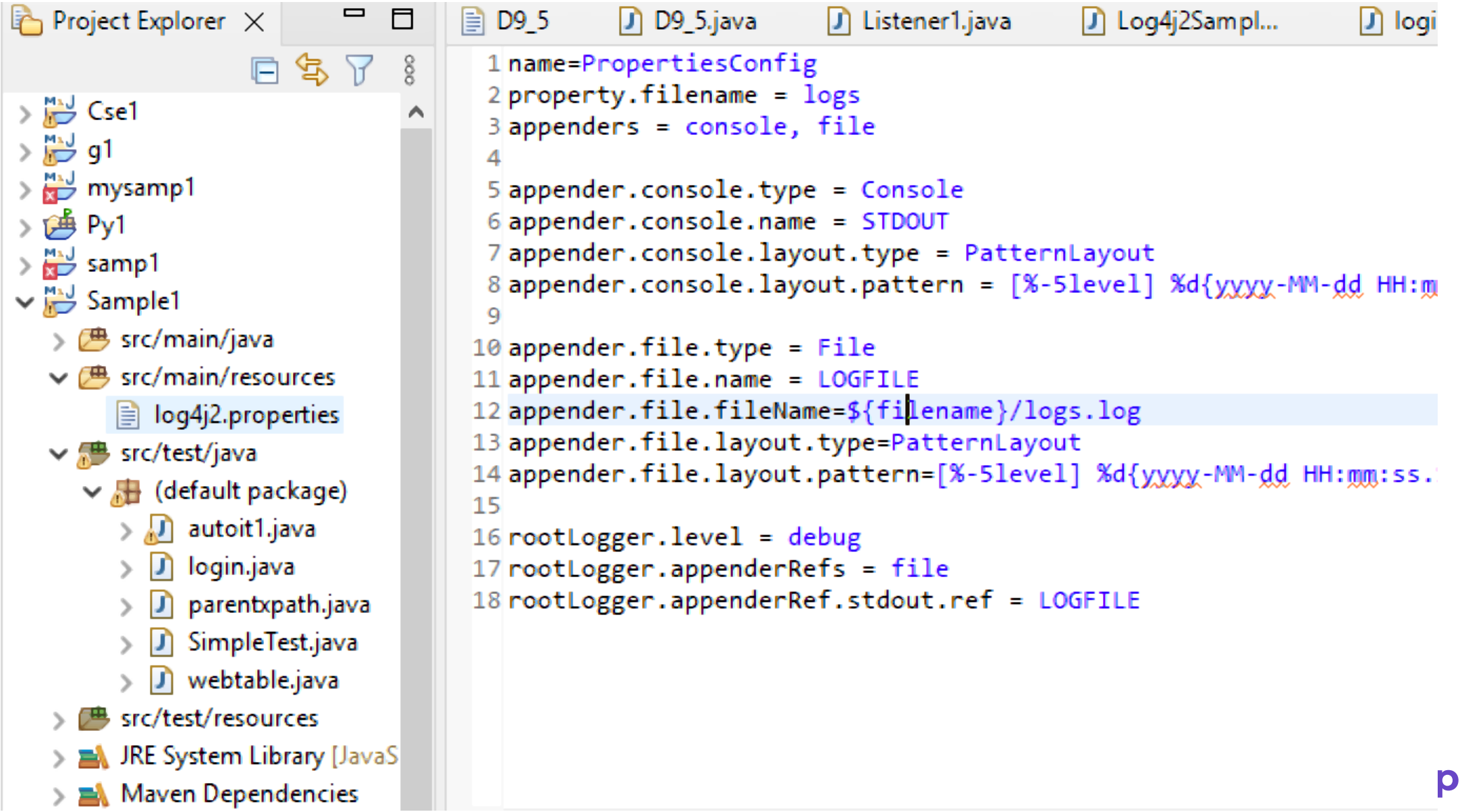
4. **Add the messages that will be displayed in the log file**

1. Write an automation script

```
public class login {
WebDriver driver;
public static Logger log; //ref variable for  Logger
@BeforeTest
public void f() {
log=LogManager.getLogger(login.class); //log instance created
//PropertyConfigurator.configure("path\\to\\log4j.properties");
WebDriverManager.chromedriver().setup();
driver=new ChromeDriver();
driver.get("https://google.com");
log.info("Openeing Google");
}
@Test
public void f1() {
String txt;
try {
txt = driver.getTitle();
System.out.println(txt);
Assert.assertEquals(txt, "Google");
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
log.error("Error Occured.. Element Not Found");
}
}
@AfterTest
public void f2() {
driver.quit();
}

}
```

2. Create log4j2.properties file in resources folder and add the following properties into it.



pleo

Create log4j2.properties file in resources folder and add the following properties into it.

```
name=PropertiesConfig
property.filename = logs
appenders = console, file


appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = [%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n


appender.file.type = File
appender.file.name = LOGFILE
appender.file.fileName=${filename}/logs.log
appender.file.layout.type=PatternLayout
appender.file.layout.pattern=[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n


rootLogger.level = debug
rootLogger.appenderRefs = file
rootLogger.appenderRef.stdout.ref = LOGFILE
```

3. messages will be displayed in the log file

eclipse-workspace - Sample1/logs/logs.log - Eclipse IDE

File Edit Navigate Search Project Pydev Run Window Help

Project Explorer

- Cse1
- g1
- mysamp1
- Py1
- samp1
- Sample1
  - src/main/java
  - src/main/resources
    - log4j2.properties
  - src/test/java
    - (default package)
      - autoit1.java
      - login.java
      - parentxpath.java
      - SimpleTest.java
      - webtable.java
  - src/test/resources
  - JRE System Library [JavaS
  - Maven Dependencies
  - Referenced Libraries
  - desktop-tutorial
  - Expleo1
  - logs
    - logs.log
  - Sample1
  - src
    - main
    - test
  - target
  - test-output
  - pom.xml

D9_5 | D9_5.java | Listener1.java | Log4j2Sampl... | login.java | log4j2.prop... | log4jmainex... | Sample1/pom.xml | logs.log

```
12 [DEBUG] 2023-09-09 12:24:24.976 [main] PlatformDependent0 - sun.misc.Unsafe.copyMemory: available
13 [DEBUG] 2023-09-09 12:24:24.976 [main] PlatformDependent0 - sun.misc.Unsafe.storeFence: available
14 [DEBUG] 2023-09-09 12:24:24.977 [main] PlatformDependent0 - java.nio.Buffer.address: available
15 [DEBUG] 2023-09-09 12:24:24.979 [main] PlatformDependent0 - direct buffer constructor: unavailable: Reflective setAccessible(true) disabled
16 [DEBUG] 2023-09-09 12:24:24.980 [main] PlatformDependent0 - java.nio.Bits.unaligned: available, true
17 [DEBUG] 2023-09-09 12:24:24.982 [main] PlatformDependent0 - jdk.internal.misc.Unsafe.allocateUninitializedArray(int): unavailable: class io.netty.util.inte
18 [DEBUG] 2023-09-09 12:24:24.984 [main] PlatformDependent0 - java.nio.DirectByteBuffer.<init>(long, int): unavailable
19 [DEBUG] 2023-09-09 12:24:24.986 [main] PlatformDependent - sun.misc.Unsafe: available
20 [DEBUG] 2023-09-09 12:24:24.987 [main] PlatformDependent - maxDirectMemory: 2128609280 bytes (maybe)
21 [DEBUG] 2023-09-09 12:24:24.988 [main] PlatformDependent - -Dio.netty.tmpdir: C:\Users\VJ\AppData\Local\Temp (java.io.tmpdir)
22 [DEBUG] 2023-09-09 12:24:24.988 [main] PlatformDependent - -Dio.netty.bitMode: 64 (sun.arch.data.model)
23 [DEBUG] 2023-09-09 12:24:24.989 [main] PlatformDependent - Platform: Windows
24 [DEBUG] 2023-09-09 12:24:24.991 [main] PlatformDependent - -Dio.netty.maxDirectMemory: -1 bytes
25 [DEBUG] 2023-09-09 12:24:24.991 [main] PlatformDependent - -Dio.netty.uninitializedArrayAllocationThreshold: -1
26 [DEBUG] 2023-09-09 12:24:24.994 [main] CleanerJava9 - java.nio.ByteBuffer.cleaner(): available
27 [DEBUG] 2023-09-09 12:24:24.994 [main] PlatformDependent - -Dio.netty.noPreferDirect: false
28 [DEBUG] 2023-09-09 12:24:25.010 [main] PlatformDependent - org.jctools-core.MpscChunkedArrayQueue: available
29 [DEBUG] 2023-09-09 12:24:25.043 [main] InternalThreadLocalMap - -Dio.netty.threadLocalMap.stringBuilder.initialSize: 1024
30 [DEBUG] 2023-09-09 12:24:25.044 [main] InternalThreadLocalMap - -Dio.netty.threadLocalMap.stringBuilder.maxSize: 4096
31 [DEBUG] 2023-09-09 12:24:25.123 [main] JdkSslContext - Default protocols (JDK): [TLSv1.3, TLSv1.2]
32 [DEBUG] 2023-09-09 12:24:25.123 [main] JdkSslContext - Default cipher suites (JDK): [TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_128_
33 [DEBUG] 2023-09-09 12:24:25.142 [main] GlobalEventExecutor - -Dio.netty.globalEventExecutor.quietPeriodSeconds: 1
34 [DEBUG] 2023-09-09 12:24:25.158 [main] MultithreadEventLoopGroup - -Dio.netty.eventLoopThreads: 8
35 [DEBUG] 2023-09-09 12:24:25.168 [main] NioEventLoop - -Dio.netty.noKeySetOptimization: false
```

Progress | Results of running class login | JUnit | Terminal | Console

```
<terminated> login [TestNG] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (09-Sep-2023, 12:25:25 pm – 12:25:40 pm) [pid: 15516]
  (Session info: chrome=116.0.5845.180)
Build info: version: '4.11.0', revision: '040bc5406b'
System info: os.name: 'Windows 10', os.arch: 'amd64', os.version: '10.0', java.version: '17.0.5'
Driver info: org.openqa.selenium.chrome.ChromeDriver
Command: [3c07df859218b5c37fd9dbf1b81087ce, getTitle {}]
Capabilities {acceptInsecureCerts: false, browserName: chrome, browserVersion: 116.0.5845.180, chrome: {chromedriverVersion: 116.0.5845.96 (1a3918166880..., userD
Session ID: 3c07df859218b5c37fd9dbf1b81087ce
        at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:77)
        at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)
```

Why Use Log4j in Selenium?

Why Use Log4j in Selenium?

- Log4j logging framework can help in debugging applications easily. With different log levels, it becomes easier to sort information based on categories.
- The logging framework is open source and can help in logging in different log levels and suppress logs in different environments – which ultimately improves application performance.
- The framework produces better output in general.
- With three components and clarity in usage, it becomes easier to use and configure log4j in Selenium.
- The setup is easy and free of cost, making it more accessible for faster debugging.