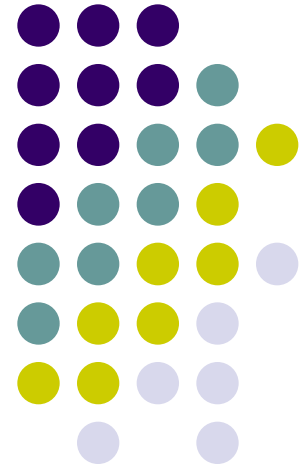


Pexus PerfLog



Easy to use Performance Logging and Diagnostic Framework for J2EE/Java Applications

- Overview
- PerfLog Internals
- PerfLog Usage
- Property Files
- Deployment Choices
- PerfLog Editions



PerfLog Overview



- ❖ What is PerfLog?
- ❖ Why PerfLog?
- ❖ PerfLog Internals
- ❖ Usage
 - ❖ Servlet Filter, Portlet Filter, Struts1, Web Service handlers, SQL
- ❖ Deployment choices
- ❖ Property files
 - ❖ Static Dynamic Tunable Properties Implementation
- ❖ PerfLog Editions
 - ❖ Community Edition (CE)
 - ❖ Supported Edition (SE) for IBM WebSphere Environment
 - ❖ Performance Data Dashboard and Visualization
 - ❖ Consulting Services for Customization
- ❖ Resources

What is PerfLog



- ❖ Performance logging and diagnosis logger framework for J2EE Applications
 - ❖ Can be used with /J2SE/Stand alone Java applications also
- ❖ Uses J2EE filter pattern for easy integration with application code
- ❖ Low or no overhead capture of key request performance metrics
 - ❖ Servlets, Portlets, Web Service (JAX-RPC, JAX-WS), JDBC SQL, Custom Transactions
 - ❖ Support for *asynchronous thread* and *JMS/Q* based logging
 - ❖ Persist performance metrics to database and log files
 - ❖ Extensible logger implementations
- ❖ Complements and enhances application logging
 - ❖ Includes PerfLogAppLogger – application logger based on `java.util.logging.*` API
 - ❖ Cache application debug trace per request and log for diagnosing slow requests.
- ❖ Support for tracking requests through multiple JVMs
- ❖ Free and Open source Community Edition and paid Supported Edition

Why PerfLog?



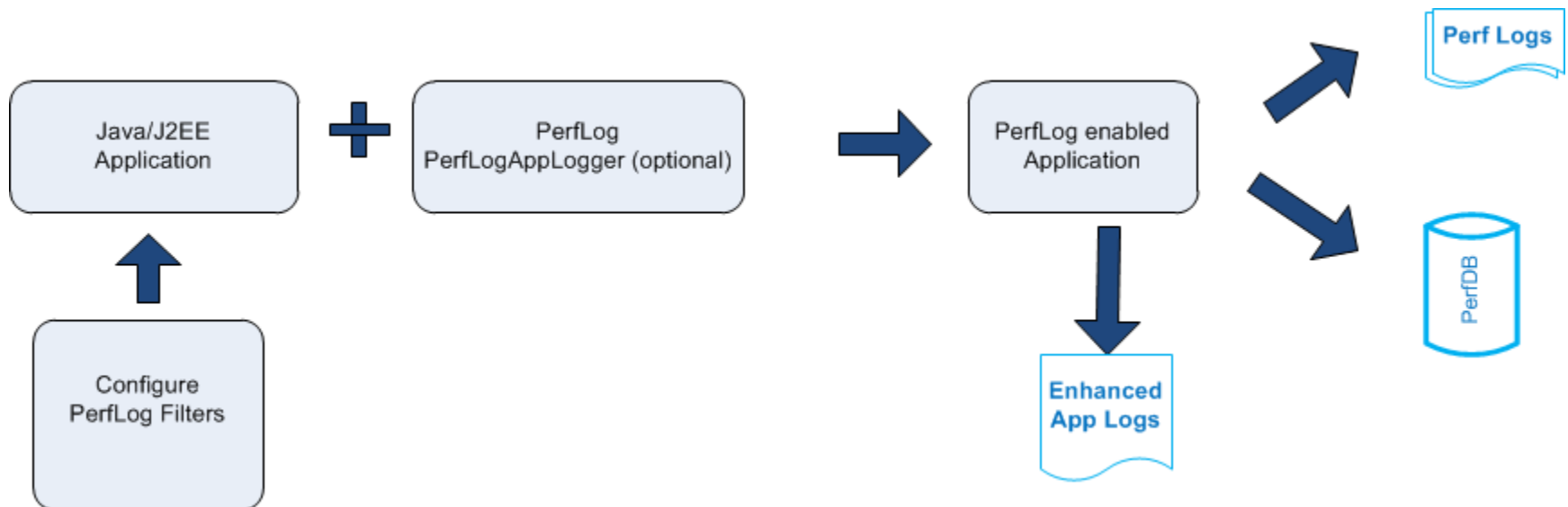
- ❖ Very low or no overhead performance logging for J2EE applications with request and application context data
- ❖ J2EE filter pattern for easy plugin-in with no application code change
 - ❖ Servlet – *web.xml*, Portlet – *portlet.xml*, web service – *web.xml*, *webservices.xml*, *handler.xml*, Struts 1 – *strutsconfig.xml*
 - ❖ JDBC SQL – *data source implementation class* in JDBC provider definition
- ❖ Track and monitor your database SQL query performance with full bind variable values for precise query parameters diagnosis
- ❖ Optionally print SOAP message request / response
- ❖ Caches debug trace statements from application logger and log only when transaction response time exceeds specified threshold
- ❖ Enhanced application logging with request and debug context for easy consumption by log monitoring and analytics tools such as Splunk
- ❖ Correlate requests/transactions spanning multiple JVM logs.
 - ❖ Very useful in large clustered environment consisting of hundreds of JVMs of various application clusters
- ❖ Sample Enhanced application logger already included or easily integrate with your existing application logger

Why PerfLog?



- ❖ Get quick answers to questions like –
 - Where did the request originate ?
 - What were the input parameters that were used by the user?
 - What variables were used in the slow performing SQL query?
 - What was SOAP message that originated from the offending request?
 - Who was the user?
 - How many JVM instances my request go to?
 - Wish I had my debug statements when this problem occur with elapsed time data for each statement !
 - How many SQLs did my web request execute?
 - Which request is the slowest and what is the frequency of such requests?

PerfLog Value Add

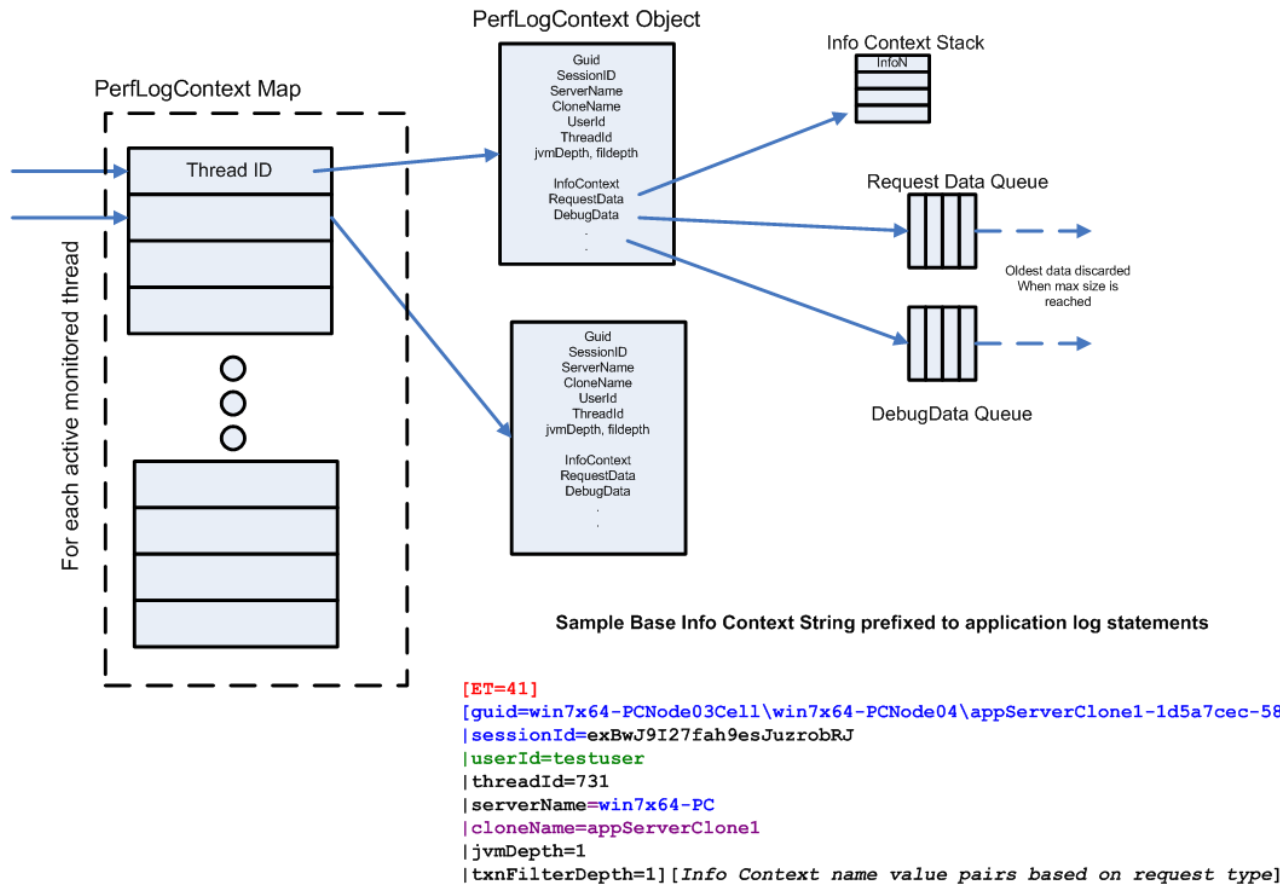




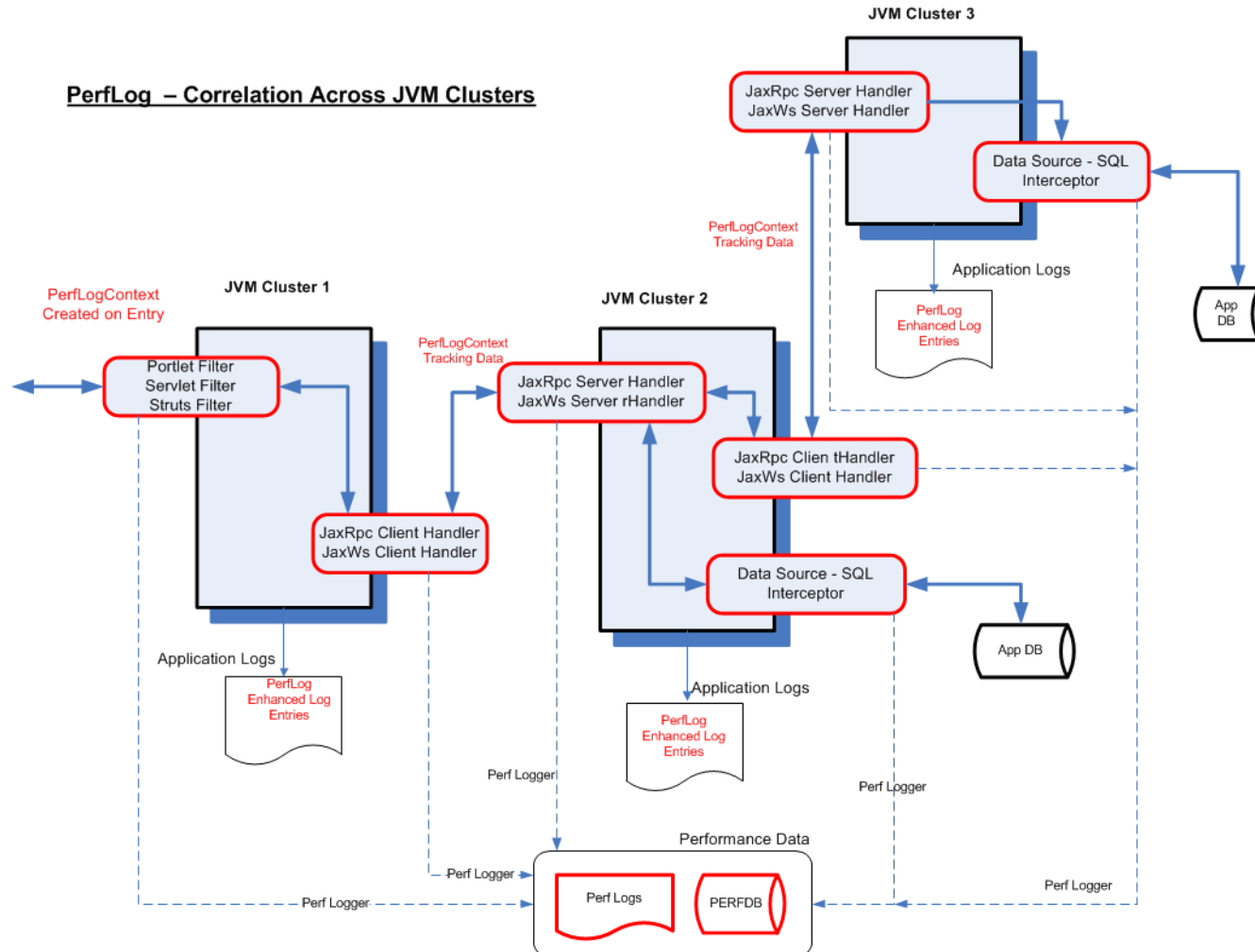
PerfLog Internals - How does PerfLog work?

- Intercept request via filters
 - Servlet filter, Portlet filter, Struts filter, Web Service handlers, SQL interceptors
- Each request is associated with a thread ID
- The first filter interception for every request creates a context called **PerfLogContext** that is mapped to the thread ID
 - Each PerfLogContext also has a globally unique identifier - called guid
 - Guid is constructed using : <JVM instance name>-Java-UUID
- Each PerfLogContext object can consists of one or more transaction filters
 - *E.g. a servlet request creates a PerfLogContext object as a result of Servlet filter interception. The same servlet request goes on to make a web service request. If this web service request has PerfLog handler associated with it; it will create a new transaction filter.*
- Cache request specific data in the PerfLogContext object
- Time the request execution at filter level and also at the global level
- Introspect request data and determine interested transactions to monitor e.g.
 - *Servlet URI, Portlet Name, Portal page, Request Parameters, Struts Action, Web Service out bound endpoint, Web Service inbound URL, SQL Query etc.*
- Cache additional data in PerfLogContext object e.g.
 - User id, Server Name, JVM instance Name, request Session ID, thread Id, etc.
 - Request Data – query parameter name, SOAP message, SQLs executed
 - Application debug trace data when current application log level is not set to debug (i.e. usually in production)
- Log request response metrics via PerfLogger interface to configured target loggers
- Check for request response time threshold and dump PerfLogContext object for diagnosis

PerfLog Internals - PerfLogContext Data Structure

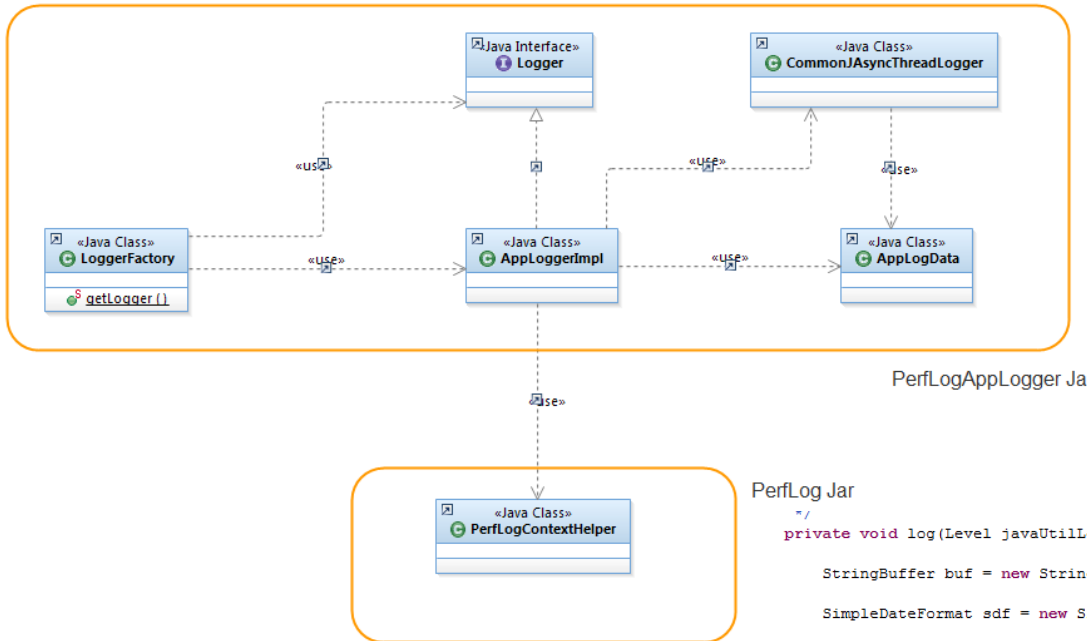


PerfLog Internals - Correlation across JVM Clusters





PerfLog Internals - Integrating PerfLog with Application Loggers



- Any application logger can be easily integrated with PerfLog using **PerfLogContextHelper** class
- Prefix **PerfLogContext** information to log statements

PerfLog Jar

```
private void log(Level javaUtilLoggerLevel, String appLoggerLevelStr, String message, Throwable t) {  
    StringBuffer buf = new StringBuffer();  
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy HH:mm:ss:SSS z");  
    String formattedTransactionDate = null;  
    formattedTransactionDate = "[" + sdf.format(new java.util.Date()) + "];"  
    buf.append(formattedTransactionDate);  
    buf.append(" ");  
    buf.append(appLoggerLevelStr);  
    buf.append(" ");  
    buf.append(loggerName);  
    buf.append(" - ");  
    buf.append(PerfLogContextHelper.getBaseAndInfoContextString());  
    buf.append(message);  
    if (javaUtilLogger.isLoggable(javaUtilLoggerLevel)) {
```

PerfLog Internals – Integrating PerfLog with Application Loggers



- Cache application log statements in PerfLogContext object, when application logger log level indicates they are not normally loggable
- Dump the PerfLogContext when response time threshold is exceeded or in case of Exception

```
private void log(Level javaUtilLoggerLevel, String appLoggerLevelStr, String message, Throwable t) {

    StringBuffer buf = new StringBuffer();

    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy HH:mm:ss:SSS z");

    String formattedTransactionDate = null;
    formattedTransactionDate = "[" + sdf.format(new java.util.Date()) + "]";

    buf.append(formattedTransactionDate);
    buf.append(" ");
    buf.append(appLoggerLevelStr);
    buf.append(" ");
    buf.append(loggerName);
    buf.append(" - ");

    buf.append(PerfLogContextHelper.getBaseAndInfoContextString());
    buf.append(message);

    if (javaUtilLogger.isLoggable(javaUtilLoggerLevel)) {

        if (isUseAsynchronousLogging()) {
            AppLogData appLogData = new AppLogData();
            appLogData.setJavaUtilLogger(javaUtilLogger);
            appLogData.setJavaUtilLoggerLevel(javaUtilLoggerLevel);
            appLogData.setLogData(buf.toString());
            appLogData.setThrowable(t);
            asyncLogger.log(appLogData);
        }
        else {
            if (t != null) {
                javaUtilLogger.log(javaUtilLoggerLevel, buf.toString(), t);
            }
            else {
                javaUtilLogger.log(javaUtilLoggerLevel, buf.toString());
            }
        }
    }
    else {
        // Cache the trace data in the debug context
        // This would be useful for diagnosis when the context data
        // is dumped when there is error or response time exceeds a defined
        // threshold
        PerfLogContextHelper.addToDebugContext("trace", buf.toString());
    }

    // Dump the context data when there is an error
    if (javaUtilLogger.isLoggable(javaUtilLoggerLevel) && javaUtilLoggerLevel == Level.SEVERE) {
        PerfLogContextHelper.dumpPerfLogContext(javaUtilLogger);
    }
}
```

Prefix PerfLog Context
Data to Log Statement

Cache application log
statement if not loggable

Dump the context when
there is an exception



Sample Enhanced Application Logger output

```
[10/07/12 16:00:49:744 CDT] INFO ServiceInvoker - [ET=725][guid=win7x64-PCNode03Cell  
\\win7x64-PCNode04\\server_9084-bb63fc1a-ce0a-42e2-8c9f-7e27c9ad2c33|  
sessionId=o1o1qYa01xcwaIVSjJEWSrv|userId=null|threadId=86|serverName=win7x64-PC|  
serverIp=10.77.14.68|cloneName=win7x64-PCNode03Cell\\win7x64-PCNode04\\server_9084|  
jvmDepth=1|txnFilterDepth=2][uri=/PerfLogTestWebApp/ServiceInvoker|host=localhost|  
port=9081|MyCustomContextName=MyCustomContextValue] lookupService: serviceName =  
service/HelloWorldServiceJaxWs
```

```
[10/07/12 17:45:56:526 CDT] INFO ServiceInvoker - [ET=1][guid=win7x64-PCNode03Cell\\win7x64  
-PCNode04\\server_9084-3d88f035-03b2-4b2b-8d9a-6164e759a232|  
sessionId=o1o1qYa01xcwaIVSjJEWSrv|userId=null|threadId=86|serverName=win7x64-PC|  
serverIp=10.77.14.68|cloneName=win7x64-PCNode03Cell\\win7x64-PCNode04\\server_9084|  
jvmDepth=1|txnFilterDepth=2][uri=/PerfLogTestWebApp/ServiceInvoker|host=localhost|  
port=9081|MyCustomContextName=MyCustomContextValue] In doGet()
```

```
[10/07/12 17:45:56:526 CDT] INFO ServiceInvoker - [ET=1][guid=win7x64-PCNode03Cell\\win7x64  
-PCNode04\\server_9084-3d88f035-03b2-4b2b-8d9a-6164e759a232|  
sessionId=o1o1qYa01xcwaIVSjJEWSrv|userId=null|threadId=86|serverName=win7x64-PC|  
serverIp=10.77.14.68|cloneName=win7x64-PCNode03Cell\\win7x64-PCNode04\\server_9084|  
jvmDepth=1|txnFilterDepth=2][uri=/PerfLogTestWebApp/ServiceInvoker|host=localhost|  
port=9081|MyCustomContextName=MyCustomContextValue] lookupService: serviceName =  
service/HelloWorldServiceJaxRpcService
```

```
[10/07/12 17:45:56:552 CDT] INFO ServiceInvoker - [ET=27][guid=win7x64-PCNode03Cell  
\\win7x64-PCNode04\\server_9084-3d88f035-03b2-4b2b-8d9a-6164e759a232|  
sessionId=o1o1qYa01xcwaIVSjJEWSrv|userId=null|threadId=86|serverName=win7x64-PC|  
serverIp=10.77.14.68|cloneName=win7x64-PCNode03Cell\\win7x64-PCNode04\\server_9084|  
jvmDepth=1|txnFilterDepth=2][uri=/PerfLogTestWebApp/ServiceInvoker|host=localhost|  
port=9081|MyCustomContextName=MyCustomContextValue] lookupService: serviceName =  
service/HelloWorldServiceJaxWs|
```

Sample PerfLogContext Dump

- PerfLog Context dump when response time exceeds specified threshold or when there is an exception
- Dump contains cached debug trace statements that are not logged normally
- Dump would also contain elapsed time for each debug trace statement with elapsed time (ET)
- Dump would also contain. JVM stats, additional request specific data and context and any custom info context added by application using *PerfLogContextHelper.pushInfoContext()*



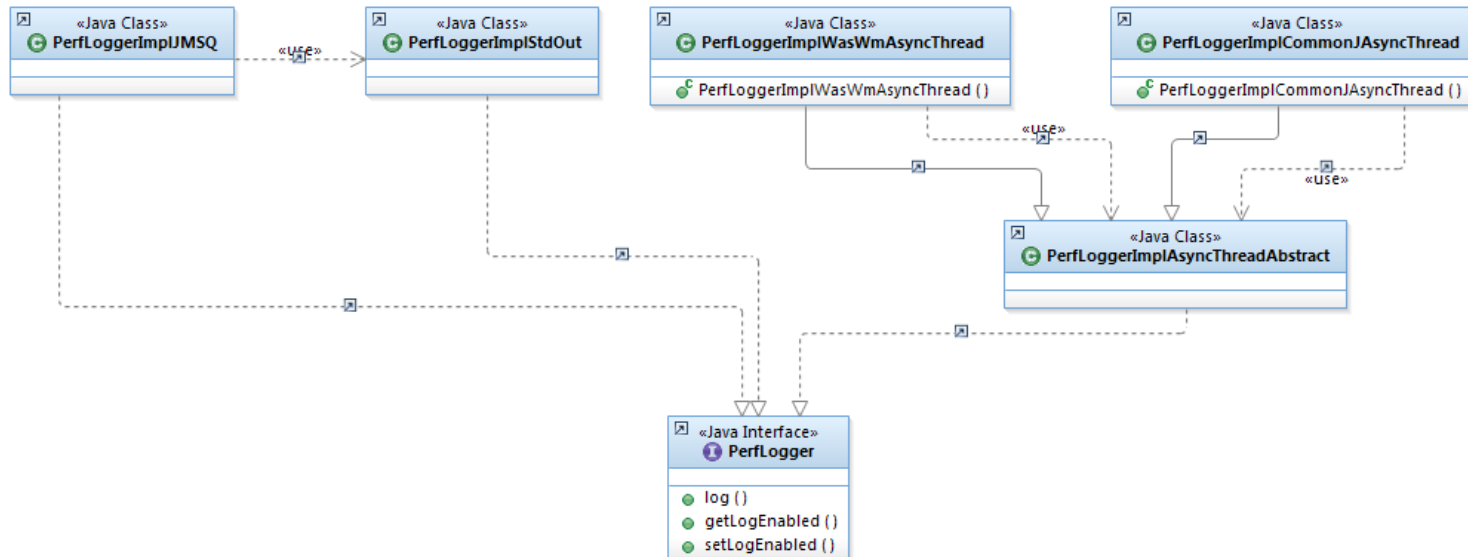
```
INFO PerfLogContextHelper - [ET=16029][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-PC|serverIp=10.77.14.68|
cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Elapsed time exceeded response time threshold: 15000
INFO PerfLogContextHelper - [ET=16038][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-PC|serverIp=10.77.14.68|
cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] : Full Request PerfLog Context Data
===== Request PerfLog Context Data Begin =====
context creation time = 10/07/12 21:54:30:711 CDT
elapsedTime since context creation (ms) = 16036
this JVM entry time = 10/07/12 21:54:30:711 CDT
elapsedTime since this JVM entry = 16036
responseTimeThreshold (ms) = 15000
txnList = []
currentDebugContextDataSize / Max Size = 4812 / 10240
currentRequestDataSize / Max Size = 0 / 10240
JVM Stats - At context creation:
  threadCount = 8
  heapMemUsage = init = 4194304(4096K) used = 4744768(4633K) committed = 5374976(5249K) max = 2147483648(2097152K)
  nonHeapMemUsage = init = 0(0K) used = 13962028(13634K) committed = 25210332(24619K) max = -1(-1K)
  GCName = MarkSweepCompact GCCount = 3 GCTime (ms) = 9
JVM Stats - Current:
  threadCount = 8
  heapMemUsage = init = 4194304(4096K) used = 4128840(4032K) committed = 6423552(6273K) max = 2147483648(2097152K)
  nonHeapMemUsage = init = 0(0K) used = 13973100(13645K) committed = 24686044(24107K) max = -1(-1K)
  GCName = MarkSweepCompact GCCount = 4 GCTime (ms) = 14
  Debug Context Data
[trace=[ET=3][10/07/12 21:54:30:713 CDT] DEBUG TestApp - [ET=3][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-PC|
serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 0
[trace=[ET=1005][10/07/12 21:54:31:715 CDT] DEBUG TestApp - [ET=1004][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 1
[trace=[ET=2007][10/07/12 21:54:32:717 CDT] DEBUG TestApp - [ET=2006][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 2
[trace=[ET=3008][10/07/12 21:54:33:719 CDT] DEBUG TestApp - [ET=3008][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 3
[trace=[ET=4010][10/07/12 21:54:34:720 CDT] DEBUG TestApp - [ET=4009][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 4
[trace=[ET=5012][10/07/12 21:54:35:722 CDT] DEBUG TestApp - [ET=5011][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 5
[trace=[ET=6014][10/07/12 21:54:36:724 CDT] DEBUG TestApp - [ET=6013][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 6
[trace=[ET=7016][10/07/12 21:54:37:726 CDT] DEBUG TestApp - [ET=7015][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 7
[trace=[ET=8018][10/07/12 21:54:38:728 CDT] DEBUG TestApp - [ET=8017][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 8
[trace=[ET=9019][10/07/12 21:54:39:729 CDT] DEBUG TestApp - [ET=9019][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|serverName=win7x64-
PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 9
[trace=[ET=10021][10/07/12 21:54:40:731 CDT] DEBUG TestApp - [ET=10020][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|
serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 10
[trace=[ET=11022][10/07/12 21:54:41:732 CDT] DEBUG TestApp - [ET=11021][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|
serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 11
[trace=[ET=12023][10/07/12 21:54:42:734 CDT] DEBUG TestApp - [ET=12023][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|
serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 12
[trace=[ET=13025][10/07/12 21:54:43:735 CDT] DEBUG TestApp - [ET=13024][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|
serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 13
[trace=[ET=14026][10/07/12 21:54:44:737 CDT] DEBUG TestApp - [ET=14026][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|
serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 14
[trace=[ET=15027][10/07/12 21:54:45:738 CDT] DEBUG TestApp - [ET=15027][guid=5876@win7x64-PC-c567315d-f30a-44df-9e62-1a550f73f655|sessionId=null|userId=null|threadId=1|
serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5876@win7x64-PC|jvmDepth=1|txnFilterDepth=1][null] Debug Statement: Sleep Count = 15]
===== Request PerfLog Context Data End =====
```


Sample PerfLogContext Dump



```
[10/13/12 21:52:00:896 CDT] 00000017 SystemOut      O INFO PerfLogContextHelper - [ET=1680][guid=5460@win7x64-PC-330a970f-6407-4cbd-b14c-3bae190c899d|sessionId=null|userId=null|threadId=76|serverName=win7x64-PC|serverIp=10.77.14.68|cloneName=5604@win7x64-PC|jvmDepth=2|txnFilterDepth=1][cJvmHost=win7x64-PC|cJvmClone=5460@win7x64-PC|javax.xml.ws.wsdl.service={http://test/HelloWorldJaxWs}HelloWorldServiceJaxWs|TransportInURL=http://localhost:9081/HelloWorldJaxWsWebServiceProject/HelloWorldServiceJaxWs] : Full Request PerfLog Context Data
===== Request PerfLog Context Data Begin =====
context creation time = 10/13/12 21:51:59:216 CDT
elapsedTime since context creation (ms) = 1680
this JVM entry time = 10/13/12 21:52:00:306 CDT
elapsedTime since this JVM entry = 590
responseTimeThreshold (ms) = 30000
txnList = [http://localhost:9081/HelloWorldJaxWsWebServiceProject/HelloWorldServiceJaxWs]
currentDebugContextDataSize / Max Size = 0 / 10240
currentRequestDataSize / Max Size = 1703 / 10240
JVM Stats - At context creation:
  threadCount = 61
  heapMemUsage = init = 52428800(51200K) used = 52539512(51308K) committed = 76890112(75088K) max = 268435456(262144K)
  nonHeapMemUsage = init = 0(0K) used = 92743988(90570K) committed = 154664668(151039K) max = -1(-1K)
  GCName = MarkSweepCompact GCCount = 28 GCTime (ms) = 980
JVM Stats - Current:
  threadCount = 61
  heapMemUsage = init = 52428800(51200K) used = 81815544(79897K) committed = 83369984(81416K) max = 268435456(262144K)
  nonHeapMemUsage = init = 0(0K) used = 96085412(93833K) committed = 158606140(154888K) max = -1(-1K)
  GCName = MarkSweepCompact GCCount = 29 GCTime (ms) = 1040
----- Request Data -----
[jaxws.message.direction=[ET=1120]jaxws.message.inbound
javax.xml.ws.wsdl.service=[ET=1120]{http://test/HelloWorldJaxWs}HelloWorldServiceJaxWs
TransportInURL=[ET=1120]http://localhost:9081/HelloWorldJaxWsWebServiceProject/HelloWorldServiceJaxWs
javax.xml.ws.wsdl.description=[ET=1120]file:/D:/Projects/PerfLog/workspace/HelloWorldJaxWsWebServiceProject/WebContent/WEB-INF/wsdl/HelloWorldService.wsdl
JaxWSServerSOAPRequest=[ET=1120]<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <soapenv:Header>
    <PerfLogContextTrackingData xmlns="uri://org.perf.log.ContextTrackingData">{"callingJvmHostId":"win7x64-PC","sessionId":null,"userId":null,"guid":"5460@win7x64-PC-330a970f-6407-4cbd-b14c-3bae190c899d","callingJvmDepthStr":"1","callingJvmCloneId":"5460@win7x64-PC","callingJvmHostIp":"10.77.14.68","createTimeInMillisStr":"1350183119216"}</PerfLogContextTrackingData>
  </soapenv:Header>
  <soapenv:Body>
    <p286:helloOperation xmlns:p286="http://test/HelloWorldJaxWs">
      <arg0>HelloWorldServiceJAXWS</arg0>
    </p286:helloOperation>
  </soapenv:Body>
</soapenv:Envelope>
JaxWSServerSOAPResponse=[ET=1670]<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:helloOperationResponse xmlns:ns2="http://test/HelloWorldJaxWs">
      <return>Sat Oct 13 21:52:00 CDT 2012: Hello - HelloWorldServiceJAXWS</return>
    </ns2:helloOperationResponse>
  </soapenv:Body>
</soapenv:Envelope>
===== Request PerfLog Context Data End =====
```

PerfLog Internals - PerfLogger - Implementations



Performance Logging Implementations – Selectable via perfLog.properties

- **PerfLoggerImplStdOut** – Logs Performance metrics to Standard Out
- **PerfLoggerImplJMSQ**
 - Logs Performance metrics to a JMS Q
 - Sample MDB provided that reads the Q and writes to log file and performance database
- **PerfLoggerImplCommonJAsyncThread**
 - Logs using asynchronous Common J Work Manager threads
- **PerfLoggerImplWasWmAsyncThread**
 - Similar to Common J Work Manager thread implementation, but using WebSphere Work Manager Thread APIs

PerfLog Internals - Logging Performance Metrics



«Java Class»
PerfLogData

- LINE_SEPARATOR: String
- logger: Logger
- guid: String
- sessionId: String
- threadName: String
- threadId: String
- transactionDate: Date
- serverName: String
- serverIp: String
- cloneName: String
- jvmDepth: int
- txnFilterDepth: int
- transactionType: String
- userId: String
- transactionName: String
- subTransactionName: String
- transactionClass: String
- transactionTime: long
- message: String
- infoContextString: String
- throwableClassName: String
- throwableMessage: String
- perfLogContext: PerfLogContext
- throwable: Throwable

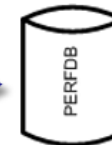
```

message=null
[09/22/12 20:24:11:080 CDT] PERFLOG(SUCCESS) :txnType=sqlQuery :txnDate=Sat Sep 22 20:24:11 CDT 2012 :txnTime=2 :userid=null :guid=win7x64-PCNode03Cell\win7x64-PCNode04\server_9084-60efa2b8-2a9f-48d5-a96b-57846598bbe8 :sessionId=null :threadName=[server.startup : 0] :threadId=null :serverName=win7x64-PC :serverIp=10.77.14.68 :cloneName=win7x64-PCNode03Cell\win7x64-PCNode04\server_9084 :jvmDepth=0 :txnFilterDepth=0 :txnName=executeQuery :subTxnName=fe388e37 # SELECT T1.INVENTORYID, T1.NAME, T1.HEADING, T1.DESRIPTION, T1.PKGINFO, T1.I... :txnClass=dbInquiryTxnClass :infoCtxStr=null :message=[dbcon=1] SELECT T1.INVENTORYID, T1.NAME, T1.HEADING, T1.DESRIPTION, T1.PKGINFO, T1.IMAGE, T1.IMGBYTES, T1.PRICE, T1.COST, T1.CATEGORY, T1.QUANTITY, T1.NOTES, T1.ISPUBLIC, T1.MINTHRESHOLD, T1.MAXTHRESHOLD FROM INVENTORY T1 WHERE T1.INVENTORYID = 'F0012' {sqlExecutionTime=2 msec}
    
```

Columns:

Primary	Name	Data Type
<input type="checkbox"/>	TXN_DT	DATE
<input type="checkbox"/>	TXN_START	TIMESTAMP
<input type="checkbox"/>	TXN_START_MS	BIGINT(20)
<input type="checkbox"/>	REQUEST_GUID	VARCHAR(255)
<input type="checkbox"/>	REQUEST_SESSION_ID	VARCHAR(255)
<input type="checkbox"/>	HOST_NAME	VARCHAR(255)
<input type="checkbox"/>	HOST_IP	VARCHAR(64)
<input checked="" type="checkbox"/>	INSTANCE_NAME	VARCHAR(255)
<input type="checkbox"/>	TXN_TIME_MS	BIGINT(20)
<input type="checkbox"/>	TXN_CLASS	VARCHAR(255)
<input type="checkbox"/>	INFO_CONTEXT	VARCHAR(2048)
<input type="checkbox"/>	TXN_TYPE	VARCHAR(255)
<input type="checkbox"/>	TXN_NAME	VARCHAR(255)
<input type="checkbox"/>	SUB_TXN_NAME	VARCHAR(255)
<input type="checkbox"/>	REQUEST_GUID_SOURCE	VARCHAR(255)
<input type="checkbox"/>	USER_ID	VARCHAR(255)
<input type="checkbox"/>	THREAD_NAME	VARCHAR(255)
<input type="checkbox"/>	TXN_STATUS	VARCHAR(255)
<input type="checkbox"/>	THREAD_ID	VARCHAR(255)
<input type="checkbox"/>	JVM_DEPTH	INT(11)
<input type="checkbox"/>	TXN_FILTER_DEPTH	INT(11)

Perf Logs



PerfLog Usage – Servlet Filter - Filter Class



PerfLog Servlet Filter Class – *org.perf.log.filter.servlet.PerfServletFilter*

- Configure Servlet Filter in web.xml – Web Deployment Descriptor configuration file
- Add filter mapping to apply the filter for desired servlet URL patterns

The screenshot shows the 'Web Application Deployment Descriptor Editor' with two tabs: '*PerfLogData.dnx' and 'web.xml'. The 'Overview' tab is active, displaying a tree view of the web application components. The 'Filter (PerfLogServletFilter)' is highlighted. To the right of the tree are buttons for 'Add...', 'Remove', 'Up', and 'Down'. The 'Details' tab is also visible, showing the following configuration:

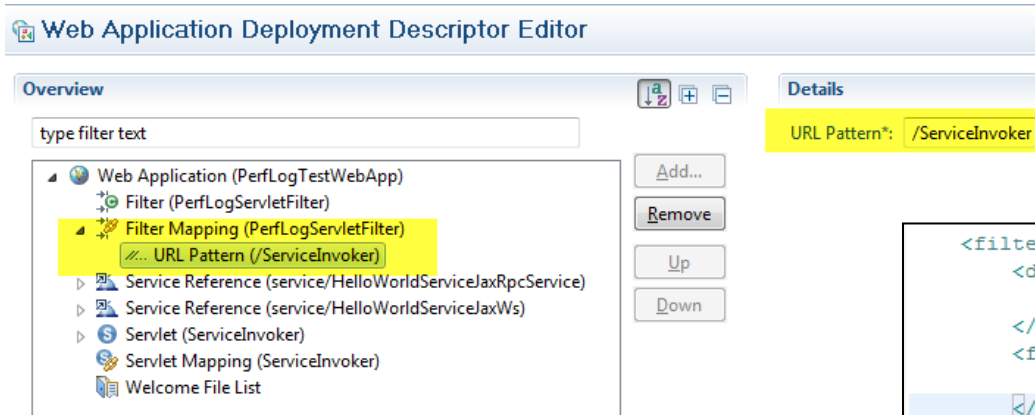
- Filter Name*: PerfLogServletFilter
- Filter Class*: org.perf.log.filter.servlet.PerfServletFilter
- Display Name: PerfLogServletFilter
- Description:
- Icon (optional):
 - Small icon:
 - Large icon:

PerfLog Usage – Servlet Filter - web.xml



Filter Mapping – *for defined Servlet Filter*

- Add URL Pattern for applying the servlet filter
- Use /* or specific servlet pattern for which the Filter will be applied
 - Add multiple URL Pattern for the same filter mapping definition



web.xml source

```
<filter>
  <display-name>
    PerfLogServletFilter
  </display-name>
  <filter-name>
    PerfLogServletFilter
  </filter-name>
  <filter-class>
    org.perf.log.filter.servlet.PerfServletFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>PerfLogServletFilter</filter-name>
  <url-pattern>/ServiceInvoker</url-pattern>
</filter-mapping>
```

PerfLog Extending *XXXPerfLogContextFilterDefaultImpl*



ServletPerfLogContextFilter.java 134 9/13/12 12:00 AM pradeep
ServletPerfLogContextFilter 134 9/13/12 12:00 AM pradeep
afterPerfLogContextCreation(ServletRequest, ServletResponse, PerfLogContext) : void
beforePerfLogContextDeletion(ServletRequest, ServletResponse, PerfLogContext, Thro
ServletPerfLogContextFilterDefaultImpl.java 143 9/16/12 7:39 AM pradeep
org.perf.log.filter.sql

```
package com.ibm.websphere.samples.plantsbywebspherewar;

import javax.servlet.ServletRequest;

public class PBWServletPerfLogContextImpl extends
    ServletPerfLogContextFilterDefaultImpl {

    public void afterPerfLogContextCreation(ServletRequest request,
        ServletResponse response, PerfLogContext perfLogContext) {

        super.afterPerfLogContextCreation(request, response, perfLogContext);
        if(request instanceof HttpServletRequest) {

            HttpServletRequest httpReq = (HttpServletRequest) request;
            HttpSession session = httpReq.getSession();
            if(session != null) {
                // get the user id and set the user id in the PerfLog context
                CustomerInfo customerInfo = (CustomerInfo)session.getAttribute(Util.ATTR_CUSTOMER);
                if(customerInfo != null) {
                    perfLogContext.setUserId(customerInfo.getCustomerID());
                }
            }
        }
    }
}
```

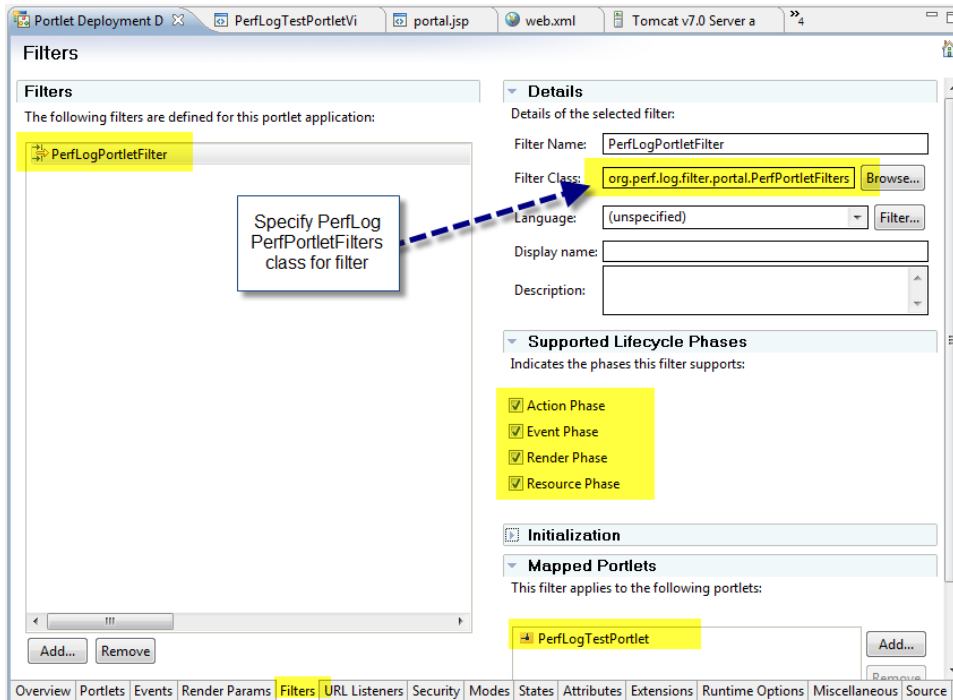
- Custom Servlet and Portlet PerfLogContext Filter can be written that extends the default implementation
- Servlet and Portlet Filter implementation implements interfaces that has the following methods that can be used to customize or extract specific context data e.g. userid from session and set it to PerfLogContext data
 - *afterPerfLogContextCreation()*
 - *beforePerfLogContextDeletion()*
- See example of extending ServletPerfLogContextFilterDefaultImpl implementation to plug-in the user id from session

PerfLog Usage – Portlet Filter - portlet.xml (WebSphere Portal)



PerfLog portlet Filter Class - *org.perf.log.filter.portal.PerfPortletFilters*

- Specify the portlet filter in portlet.xml file or use the associated editor to add a portlet filter
 - Select the phases the filter should apply to. All phases can be supported by the PerfLog portlet filter.
 - Add Portlets for the filter



portlet.xml source

```
</portlet>
<filter>
  <filter-name>PerfLogPortletFilter</filter-name>
  <filter-class>org.perf.log.filter.portal.PerfPortletFilters</filter-class>
  <lifecycle>ACTION_PHASE</lifecycle>
  <lifecycle>EVENT_PHASE</lifecycle>
  <lifecycle>RENDER_PHASE</lifecycle>
  <lifecycle>RESOURCE_PHASE</lifecycle>
</filter>
<filter-mapping>
  <filter-name>PerfLogPortletFilter</filter-name>
  <portlet-name>PerfLogTestPortlet</portlet-name>
</filter-mapping>
<default-namespace>http://PerfLogTestPortlet/</default-namespace>
```

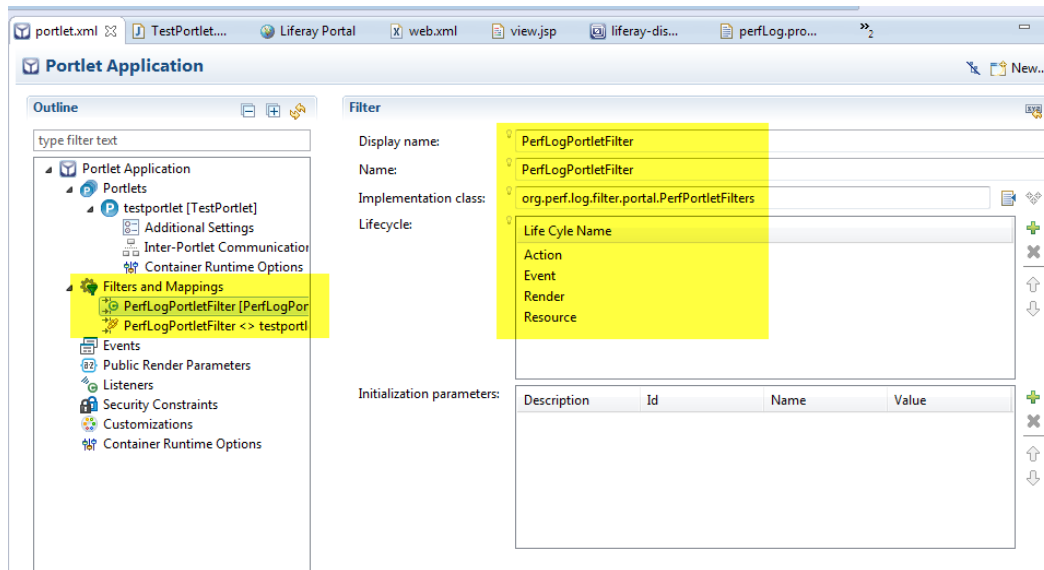
PerfLog Usage – Portlet Filter - portlet.xml (Liferay Portal)

PerfLog portlet Filter Class - *org.perf.log.filter.portal.PerfPortletFilters*

- Specify the portlet filter in portlet.xml file or use the associated editor to add a portlet filter
- Select the phases the filter should apply to. All phases can be supported by the PerfLog portlet filter.
- Add Portlets for the defined filter.



portlet.xml source



```
<?xml version="1.0"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd" xmlns:xsi="http://
<portlet>
  <portlet-name>testportlet</portlet-name>
  <display-name>TestPortlet</display-name>
  <portlet-class>com.test.TestPortlet</portlet-class>
  <init-param>
    <name>view-template</name>
    <value>/html/testportlet/view.jsp</value>
  </init-param>
  <expiration-cache>0</expiration-cache>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>view</portlet-mode>
  </supports>
  <portlet-info>
    <title>TestPortlet</title>
    <short-title>TestPortlet</short-title>
    <keywords></keywords>
  </portlet-info>
  <security-role-ref>
    <role-name>administrator</role-name>
  </security-role-ref>
  <security-role-ref>
    <role-name>guest</role-name>
  </security-role-ref>
  <security-role-ref>
    <role-name>power-user</role-name>
  </security-role-ref>
  <security-role-ref>
    <role-name>user</role-name>
  </security-role-ref>
</portlet>
<filter>
  <display-name>PerfLogPortletFilter</display-name>
  <filter-name>PerfLogPortletFilter</filter-name>
  <filter-class>org.perf.log.filter.portal.PerfPortletFilters</filter-class>
  <lifecycle>ACTION_PHASE</lifecycle>
  <lifecycle>EVENT_PHASE</lifecycle>
  <lifecycle>RENDER_PHASE</lifecycle>
  <lifecycle>RESOURCE_PHASE</lifecycle>
</filter>
<filter-mapping>
  <filter-name>PerfLogPortletFilter</filter-name>
  <portlet-name>testportlet</portlet-name>
</filter-mapping>
</portlet-app>
```



PerfLog Usage – Struts1 Filter – Struts Configuration file (WebSphere Portal)

PerfLog struts 1 filter class `org.perf.log.filter.wp.struts1.PerfStrutsRequestProcessor`

- Specify struts filter class in the Struts Configuration file

The Struts filter currently works in WebSphere Portal for Struts Portlets only.

The screenshot shows the WebSphere Admin Console interface for configuring Struts. The top navigation bar includes tabs for 'web.xml', 'Web Services Editor', 'webservice.xml', 'Admin Console', and 'Struts Configuration'. The 'Struts Configuration' tab is active, showing a 'Controller' configuration page. The 'Controller attributes' section is expanded, displaying various configuration fields. The 'Processor Class' field is highlighted in yellow and contains the text 'org.perf.log.filter.wp.struts1.PerfStrutsRequestProcessor'. Other fields include 'Buffer Size', 'Multi Part Class', 'Content Type', 'Forward Pattern', 'Maximum File Size', 'Memory File Size', 'Page Pattern', and 'Temp Directory'. The bottom navigation bar has several tabs, with 'Controller' highlighted in yellow.

Controller attributes	
Attributes of the selected Controller	
Processor Class:	<input type="text" value="org.perf.log.filter.wp.struts1.PerfStrutsRequestProcessor"/> Browse...
Buffer Size:	<input type="text"/>
Multi Part Class:	<input type="text"/> Browse...
Content Type:	<input type="text"/>
Forward Pattern:	<input type="text"/>
Maximum File Size:	<input type="text"/>
Memory File Size:	<input type="text"/>
Page Pattern:	<input type="text"/>
Temp Directory:	<input type="text"/> Browse...
<input type="checkbox"/> Input Forward	
<input type="checkbox"/> Local	

Navigation tabs: Action Mappings | Global Forwards | Form Beans | Data Sources | Global Exceptions | **Controller** | Message Resources | Plug-ins | Source

PerfLog Usage – Web Service - JAX-RPC Server Handler

PerfLog JAX-RPC Server handler class - *org.perf.log.filter.ws.JaxRpcLogContextServerHandler*

- PerfLog JAX-RPC server handler is configured in **webservices.xml** file in the services project

The screenshot displays the Eclipse IDE interface. On the left, the 'Web Services' project is open, showing a tree view with 'Web Service Overview', 'Web Service Description', 'Port Component', 'Handler', and 'Service Implementation'. The 'Handler' component is selected, and its details are shown in the 'Handler Details' panel. The 'Handler Class' is set to `org.perf.log.filter.ws.JaxRpcLogContextServerHandler`. A red arrow points from a text box labeled 'Specify PerfLog JAX-RPC Server Handler Class here' to the 'Handler Class' field.

On the right, the 'webservices.xml' source file is open. The XML content is as follows:

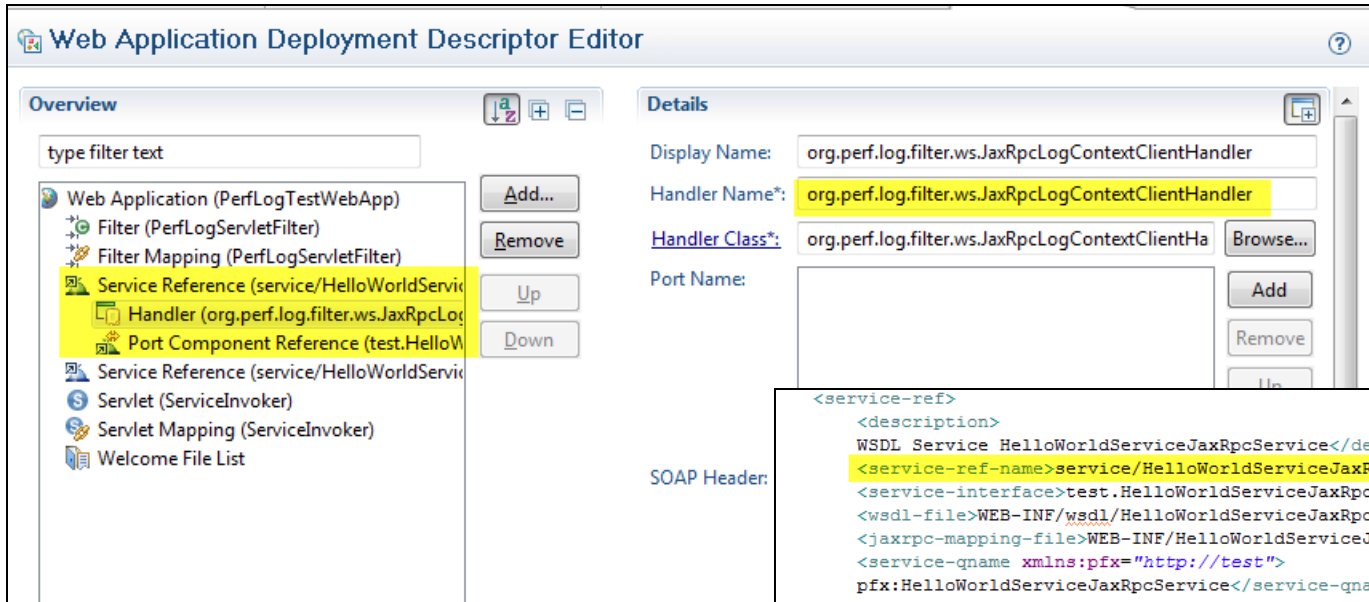
```
<?xml version="1.0" encoding="UTF-8"?>
<webservices xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-services.xsd">
    <web-service-description>
        <web-service-description-name>HelloWorldServiceJaxRpcService</web-service-description-name>
        <wsdl-file>WEB-INF/wsdl/HelloWorldServiceJaxRpc.wsdl</wsdl-file>
        <jaxrpc-mapping-file>WEB-INF/HelloWorldServiceJaxRpc_mapping.xml</jaxrpc-mapping-file>
        <port-component>
            <port-component-name>HelloWorldServiceJaxRpc</port-component-name>
            <wsdl-port xmlns:pf="http://test">
                pf:HelloWorldServiceJaxRpc</wsdl-port>
            <enable-mtom>false</enable-mtom>
            <service-endpoint-interface>test.HelloWorld</service-endpoint-interface>
            <service-impl-bean>
                <servlet-link>
                    test_HelloWorld</servlet-link>
                </service-impl-bean>
            <handler>
                <display-name>org.perf.log.filter.ws.JaxRpcLogContextServerHandler</display-name>
                <handler-name>
                    org.perf.log.filter.ws.JaxRpcLogContextServerHandler
                </handler-name>
                <handler-class>
                    org.perf.log.filter.ws.JaxRpcLogContextServerHandler
                </handler-class>
            </handler>
        </port-component>
    </web-service-description>
</webservices>
```

A red arrow points from the text box 'Specify PerfLog JAX-RPC Server Handler Class here' to the `org.perf.log.filter.ws.JaxRpcLogContextServerHandler` line in the XML file.

PerfLog Usage – Web Service - JAX-RPC Client Handler

JAX-RPC Client Handler class is *org.perf.log.filter.ws.JaxRpcLogContextClientHandler*

- Specify PerfLog JAX-RPC Client Handler class for the service reference in web.xml or ejb-jar.xml



web.xml source

- Configuration in ejb-jar.xml is similar
- ejb-jar.xml is used when an EJB calls a web service and a service reference is defined on the EJB

```
<service-ref>
  <description>
    WSDL Service HelloWorldServiceJaxRpcService</description>
    <service-ref-name>service/HelloWorldServiceJaxRpcService</service-ref-name>
    <service-interface>test.HelloWorldServiceJaxRpcService</service-interface>
    <wsdl-file>WEB-INF/wsdl/HelloWorldServiceJaxRpc.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/HelloWorldServiceJaxRpc_mapping.xml</jaxrpc-mapping-file>
    <service-qname xmlns:pfx="http://test">
      pfx:HelloWorldServiceJaxRpcService</service-qname>
    <port-component-ref>
      <service-endpoint-interface>test.HelloWorld</service-endpoint-interface>
    </port-component-ref>
    <handler>
      <display-name>
        org.perf.log.filter.ws.JaxRpcLogContextClientHandler
      </display-name>
      <handler-name>
        org.perf.log.filter.ws.JaxRpcLogContextClientHandler
      </handler-name>
      <handler-class>
        org.perf.log.filter.ws.JaxRpcLogContextClientHandler
      </handler-class>
    </handler>
  </service-ref>
```




PerfLog Usage – Web Service - JAX-RPC Client Handler

- PerfLog JAX-RPC Client Handler can also be specified programmatically
- This is useful for legacy code where service reference is not available or defined in the web.xml or ejb-jar.xml deployment description file, instead JAX-RPC service is invoked by service locator class
- Register a handler using service locator
- Helper method to register handler is available in class *org.perf.log.filter.ws.JaxRpcLogContextHandler*
- See example code below:

```
public static MyService getMyService()
{
    MyServiceServiceLocator locator = new MyServiceServiceLocator();
    try {
        // get my service endpoint from property file
        String endPoint = getServiceEndpoint(MY_SERVICE_URL_KEY);
        logger.debug("MyService endpoint: " + endPoint);
        JaxRpcLogContextHandler.registerHandler(
            locator.getHandlerRegistry(),
            locator.getServiceName(),
            JaxRpcLogContextClientHandler.class);
        // return service with registered handler
        return locator.getMyService(
            new URL(endpoint);
        } catch (Exception e) {
            logger.error("MyService", e);
        }
    }
    return null;
}
```

PerfLog Usage – Web Service - JAX-WS Server Handler



JAX-WS Server handler class is - *org.perf.log.filter.ws.JaxWSLogContextServerHandler*

- PerfLog JAX-WS server handler can be configured in webservices.xml or via handler-chain.xml file
 - Use JAX-WS annotation - @HandlerChain to refer to the handler-chain.xml file

*Sample Java Source for Web Service
Annotation @HandlerChain*

```
import javax.ws.HandlerChain;  
import javax.ws.WebMethod;  
import javax.ws.WebService;  
  
@WebService  
@HandlerChain(file="handler-chain.xml")  
public class ServerInfo{  
  
    @WebMethod  
    public String getServerName() {
```

Specifying PerfLog JAX-WS Server Handler via
@HandlerChain annotation

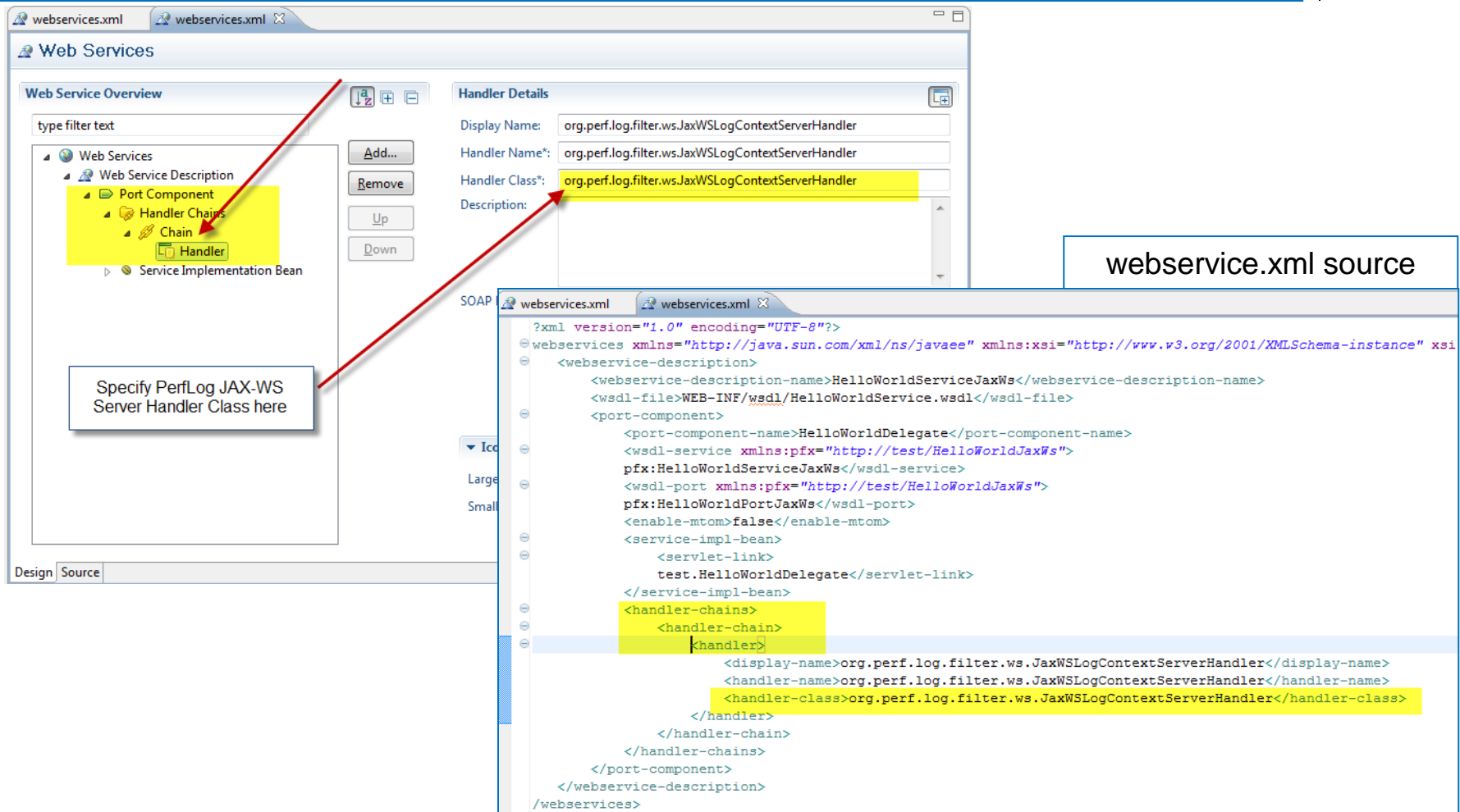
*handler-chain.xml file – this file should be in the same directory
as the service source class or use relative file path*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<javaee:handler-chains  
  xmlns:javaee="http://java.sun.com/xml/ns/javaee"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <javaee:handler-chain>  
    <javaee:handler>  
      <javaee:handler-class>org.perf.log.filter.ws.JaxWSLogContextServerHandler</javaee:handler-class>  
    </javaee:handler>  
  </javaee:handler-chain>  
</javaee:handler-chains>
```

PerfLog Usage – Web Service - JAX-WS Server Handler

JAX-WS Server handler class is - *org.perf.log.filter.ws.JaxWSLogContextServerHandler*

- Alternate way of specifying handler in webservicexml file in the services project



The screenshot displays the Eclipse IDE interface for configuring a web service. The 'Web Services' perspective is active, showing the 'Web Service Overview' on the left and the 'Handler Details' on the right. The 'Web Service Overview' shows a tree structure with 'Web Services' expanded, containing 'Web Service Description', 'Port Component', 'Handler Chains', and 'Chain'. The 'Handler' is selected under 'Chain'. The 'Handler Details' panel shows the following information:

- Display Name: `org.perf.log.filter.ws.JaxWSLogContextServerHandler`
- Handler Name: `org.perf.log.filter.ws.JaxWSLogContextServerHandler`
- Handler Class: `org.perf.log.filter.ws.JaxWSLogContextServerHandler`
- Description:

A red arrow points from the 'Handler' in the 'Chain' to the 'Handler Class' field in the 'Handler Details' panel. A text box with the text 'Specify PerfLog JAX-WS Server Handler Class here' is positioned near the arrow.

The 'webservice.xml source' panel shows the XML configuration for the web service. The XML is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<webservicexml xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/webservicexml.xsd">
  <web-service-description-name>HelloWorldServiceJaxWs</web-service-description-name>
  <wsdl-file>WEB-INF/wsdl/HelloWorldService.wsdl</wsdl-file>
  <port-component>
    <port-component-name>HelloWorldDelegate</port-component-name>
    <wsdl-service xmlns:pfx="http://test/HelloWorldJaxWs">
      pfx:HelloWorldServiceJaxWs</wsdl-service>
    <wsdl-port xmlns:pfx="http://test/HelloWorldJaxWs">
      pfx:HelloWorldPortJaxWs</wsdl-port>
    <enable-mtom>false</enable-mtom>
    <service-impl-bean>
      <servlet-link>
        test.HelloWorldDelegate</servlet-link>
      </service-impl-bean>
      <handler-chains>
        <handler-chain>
          <handler>
            <display-name>org.perf.log.filter.ws.JaxWSLogContextServerHandler</display-name>
            <handler-name>org.perf.log.filter.ws.JaxWSLogContextServerHandler</handler-name>
            <handler-class>org.perf.log.filter.ws.JaxWSLogContextServerHandler</handler-class>
          </handler>
        </handler-chain>
      </handler-chains>
    </port-component>
  </web-service-description>
</webservicexml>
```

PerfLog Usage – Web Service - JAX-WS Client Handler



JAX-WS Client handler class is - *org.perf.log.filter.ws.JaxWSLogContextClientHandler*

- PerfLog JAX-WS client handler can be configured in web.xml or via handler-chain.xml file
 - Use JAX-WS annotation - @HandlerChain to refer to the handler-chain.xml file in Service Interface

Specifying PerfLog JAX-WS Client Handler via
@HandlerChain annotation

*Sample Java Source for Web Service
Annotation @HandlerChain*

```
// Generated By:JAX-WS RI IBM 2.1.6 in JDK 6 (JAXB RI IBM JAXB 2.1.10 in JDK 6)
```

```
package testjaxws;
```

```
import java.util.List;
```

```
@WebService(name = "HelloWorldDelegate", targetNamespace = "http://testjaxws/")
```

```
@HandlerChain (file="handler-chain.xml")
```

```
@XmlSeeAlso({  
    ObjectFactory.class  
})
```

```
public interface HelloWorldDelegate {
```

```
/**
```

```
 * @param arg0  
 * @return
```

```
 *     returns java.lang.String  
 */
```

```
@WebMethod
```

```
@WebResult(targetNamespace = "")
```

```
@RequestWrapper(localName = "helloOperation", targetNamespace = "http://testjaxws/")
```

```
@ResponseWrapper(localName = "helloOperationResponse", targetNamespace = "http://testjaxws/")
```

*handler-chain.xml file – this file should be in the same directory
as the service interface or use relative path*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<javaee:handler-chains  
    xmlns:javaee="http://java.sun.com/xml/ns/javaee"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <javaee:handler-chain>  
    <javaee:handler>  
      <javaee:handler-class>  
        org.perf.log.filter.ws.JaxWSLogContextClientHandler  
      </javaee:handler-class>  
    </javaee:handler>  
  </javaee:handler-chain>  
</javaee:handler-chains>
```

PerfLog Internals – Sample SOAP Message with PerfLog tracking data



```
<?xml version="1.0" encoding="UTF-8"?>
- <soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  - <soapenv:Header>
    <PerfLogContextTrackingData xmlns="uri://org.perf.log.ContextTrackingData">
      {"callingJvmHostId":"win7x64-PC","sessionId":null,"userid":null,"guid":"5460@win7x64-PC-330a970f-6407-4cbd-b14c-3bae190c899d","callingJvmDepthStr":"1","callingJvmCloneId":"5460@win7x64-PC","callingJvmHostIp":"10.77.14.68","createTimeInMillisStr":"1350183119216"}
    </PerfLogContextTrackingData>
  </soapenv:Header>
  - <soapenv:Body>
    - <p286:helloOperation xmlns:p286="http://test/HelloWorldJaxWs">
      <arg0>HelloWorldServiceJAXWS:</arg0>
    </p286:helloOperation>
  </soapenv:Body>
</soapenv:Envelope>
```



PerfLog Usage – SQL - Interceptor

SQL Interceptor Classes - *org.perf.log.filter.sql.**

- Interceptor classes based on JDBC Data source implementation class (XA and non-XA)
- Interceptor classes included in PerfLog – DB2, MySQL, Oracle, Derby
- Additional interceptors for any JDBC driver can be easily implemented using the same pattern

Example Config: Derby JDBC Driver for IBM WebSphere Environment

- Copy PerfLog.jar to the same directory that has JDBC driver -
- Update the classpath for the JDBC provider for the application data source to include PerfLog.jar location
- Update the data source implementation class in the JDBC provider definition. Use
 - `org.perf.log.filter.sql.DerbyConnectionPoolDataSourceInterceptor` - for non-XA
 - `org.perf.log.filter.sql.DerbyConnectionPoolDataSourceXAInterceptor` - for XA

*Follow same procedure for other JDBC driver classes i.e. DB2, Oracle, MySQL
The configuration steps may vary depending on your J2EE environment.*

JDBC providers > Derby JDBC Provider (XA)

Use this page to edit properties of a Java Database Connectivity (JDBC) provider that encapsulates the specific JDBC driver implementation class for access to the environment.

Configuration

General Properties

* Scope
cells:win7x64-PCNode03Cell

* Name
Derby JDBC Provider (XA)

Description
Derby embedded XA JDBC Provider. This provider is only configurable in version 6.0.2 and later nodes

Class path
\${DERBY_JDBC_DRIVER_PATH}/derby.jar
C:\PerfLogJar\PerfLog.jar

Native library path

☐ Isolate this resource provider

* Implementation class name
org.perf.log.filter.sql.DerbyConnectionPoolDataSourceXA

Apply OK Reset Cancel

PerfLog Usage – SQL – Interceptor- Sample SQLs in logs



```
[10/13/12 22:35:09:017 CDT] 00000020 SystemOut      O PERFLOG(SUCCESS) :txnType=sqlQuery :txnDate=Sat Oct
13 22:35:09 CDT 2012 :txnTime=1 :userId=null :guid=888@win7x64-PC-2e6db485-01e4-4bc6-8a82-1dfab0cd1586
:sessionId=VzbW5UsOkTSiYXc56DBLcsw :threadName=[WebContainer : 6] :threadId=96 :serverName=win7x64-PC
:serverIp=10.77.14.68 :cloneName=888@win7x64-PC :jvmDepth=1 :txnFilterDepth=1 :txnName=executeQuery
:subTxnName=161c6347 # SELECT T1.INVENTORYID, T1.NAME, T1.HEADING, T1.DESCRPTION, T1.PKGINFO, T1.I...
:txnClass=dbInquiryTxnClass

:infoCtxStr=[uri=/PlantsByWebSphereAjax/servlet/RPCAdapter/httprpc/Sample/detailRequest|p0=T0003]
:message=[dbcon=1] SELECT T1.INVENTORYID, T1.NAME, T1.HEADING, T1.DESCRPTION, T1.PKGINFO, T1.IMAGE,
T1.IMGBYTES, T1.PRICE, T1.COST, T1.CATEGORY, T1.QUANTITY, T1.NOTES, T1.ISPUBLIC, T1.MINTHRESHOLD,
T1.MAXTHRESHOLD FROM INVENTORY T1 WHERE T1.INVENTORYID = 'T0003' {sqlExecutionTime=1 msec}
[10/13/12 22:35:09:037 CDT] 00000020 SystemOut      O PERFLOG(SUCCESS) :txnType=servlet :txnDate=Sat Oct 13
22:35:09 CDT 2012 :txnTime=30 :userId=null :guid=888@win7x64-PC-2e6db485-01e4-4bc6-8a82-1dfab0cd1586
:sessionId=VzbW5UsOkTSiYXc56DBLcsw :threadName=[WebContainer : 6] :threadId=96 :serverName=win7x64-PC
:serverIp=10.77.14.68 :cloneName=888@win7x64-PC :jvmDepth=1 :txnFilterDepth=1
:txnName=/PlantsByWebSphereAjax/servlet/RPCAdapter/httprpc/Sample/detailRequest :subTxnName=null
:txnClass=webUITxnClass

:infoCtxStr=[uri=/PlantsByWebSphereAjax/servlet/RPCAdapter/httprpc/Sample/detailRequest|p0=T0003]
:message=null
```



PerfLog Usage – Custom Transaction Monitoring

- PerfLog Can be used to time custom transactions and log using the same PerfLogger implementation
- It can also be used to add additional business context data to log statements
- See example in Sample – TestApp.java in PerfLogTestJavaApp project

```
public static void main(String[] args) {
    appLogger.info("In TestApp.main()");
    // Start a PerfLog Transaction Monitor
    TxnData txnData = new TxnData("MyTxnName1", "MySubTxnName1",
        "MyCustomTxnClass1", "MyCustomTxnType1");

    // Start monitoring the above transaction from this point..
    PerfLogContextHelper.startPerfLogTxnMonitor(txnData);
    appLogger.info("Start PerfLog Transaction Monitor..., Log statement should show additional transaction context like guid etc.");
    // Push some application context data as name / value pairs to the PerfLog Context
    PerfLogContextHelper.pushInfoContext("myApplicationContextName1", "Value1");
    PerfLogContextHelper.pushInfoContext("myApplicationContextDataName2", "Value2");
    appLogger.info("See additional Info Context data indicating the two new name/value pairs in this log statement");
    //Pop the last pushed info context
    PerfLogContextHelper.popInfoContext();
    appLogger.info("The info context for this log statement shows one of the application context data removed");
    sleepFunction(10);
    appLogger.info("End PerfLog Transaction Monitor...");
    // Log performance metrics when ending the Txn Monitor
    PerfLogContextHelper.endPerfLogTxnMonitor(true);

    // Second transaction, this time sleep for 20 seconds
    // The PerfLogContext response time threshold is set to 15 seconds
    // So this time PerfLog will also dump the PerfLog Context data
    // that contains additional context details for diagnosis in addition
    // logging performance metrics for the transaction...

    txnData = new TxnData("MyTxnName2", "MySubTxnName2",
        "MyCustomTxnClass2", "MyCustomTxnType2");
    PerfLogContextHelper.startPerfLogTxnMonitor(txnData);
    sleepFunction(16);
    PerfLogContextHelper.endPerfLogTxnMonitor(true);
}
```

Create a custom transaction Data and start the PerfLog Txn

Push additional application context data to current log context that will be logged with application log statements and also to perf log data

End the transaction and log



PerfLog Usage – Summary – PerfLog Enabling your application

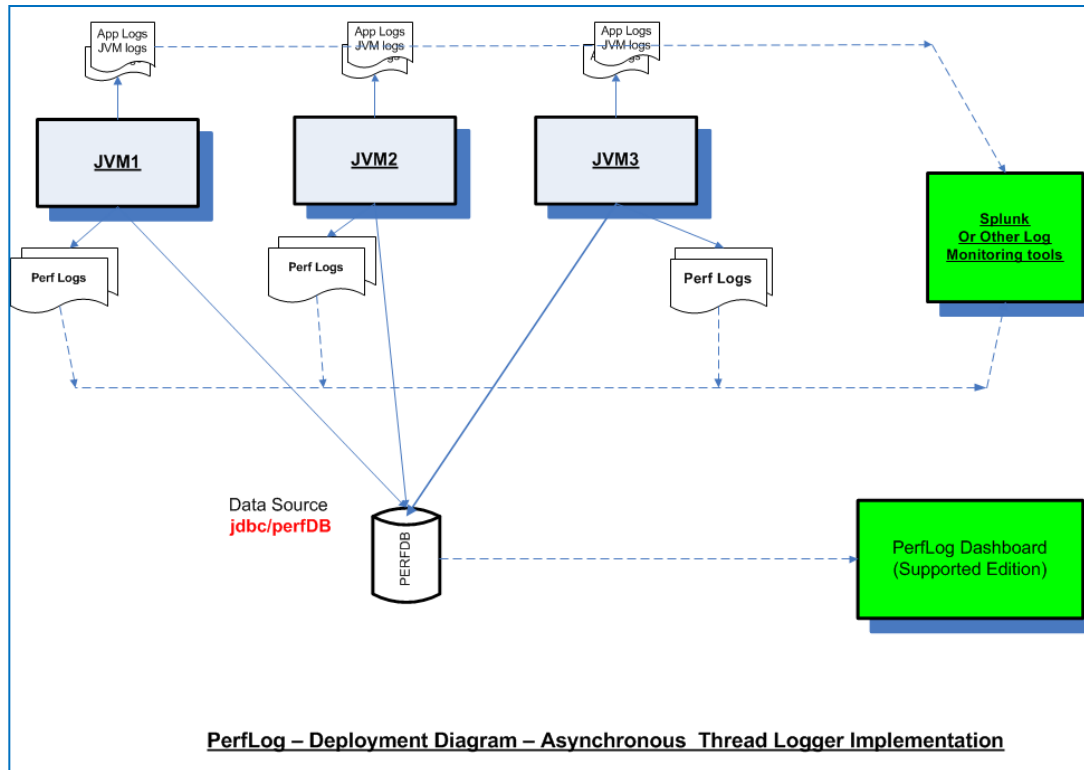
1. Include PerfLog.jar in build path of your application project
 - Optionally include PerfLogAppLogger.jar or enable your application logger to use PerfLog for enhanced logging
2. If application has Portlets, configure Portlet filter for your portlets in portlet.xml
3. If application has Servlets, configure Servlet filter for your servlets in web.xml
4. If application uses Struts (WebSphere Portal), configure Struts request processor for Struts project
5. If application uses web service as a client, create Service Reference for client side web service calls
 - Add handler class to each client service reference
6. If application includes service provider project, configure server side web service handler in services project
 - Add handler for each exposed service in the services project in webservices.xml
7. Configure application JDBC provider to include PerfLog.jar in classpath and specify the appropriate Datasource interceptor implementation instead of the default datasource implementation for the JDBC provider.
8. Include perfLog.properties and runtimeEnv.properties in your application or your application classpath to override properties that affect your environment and your choice of PerfLogger implementation, log file names, response time thresholds etc
 - Optionally include perfLogAppLogger.properties to override PerfLogAppLogger default properties..

PerfLog Deployment



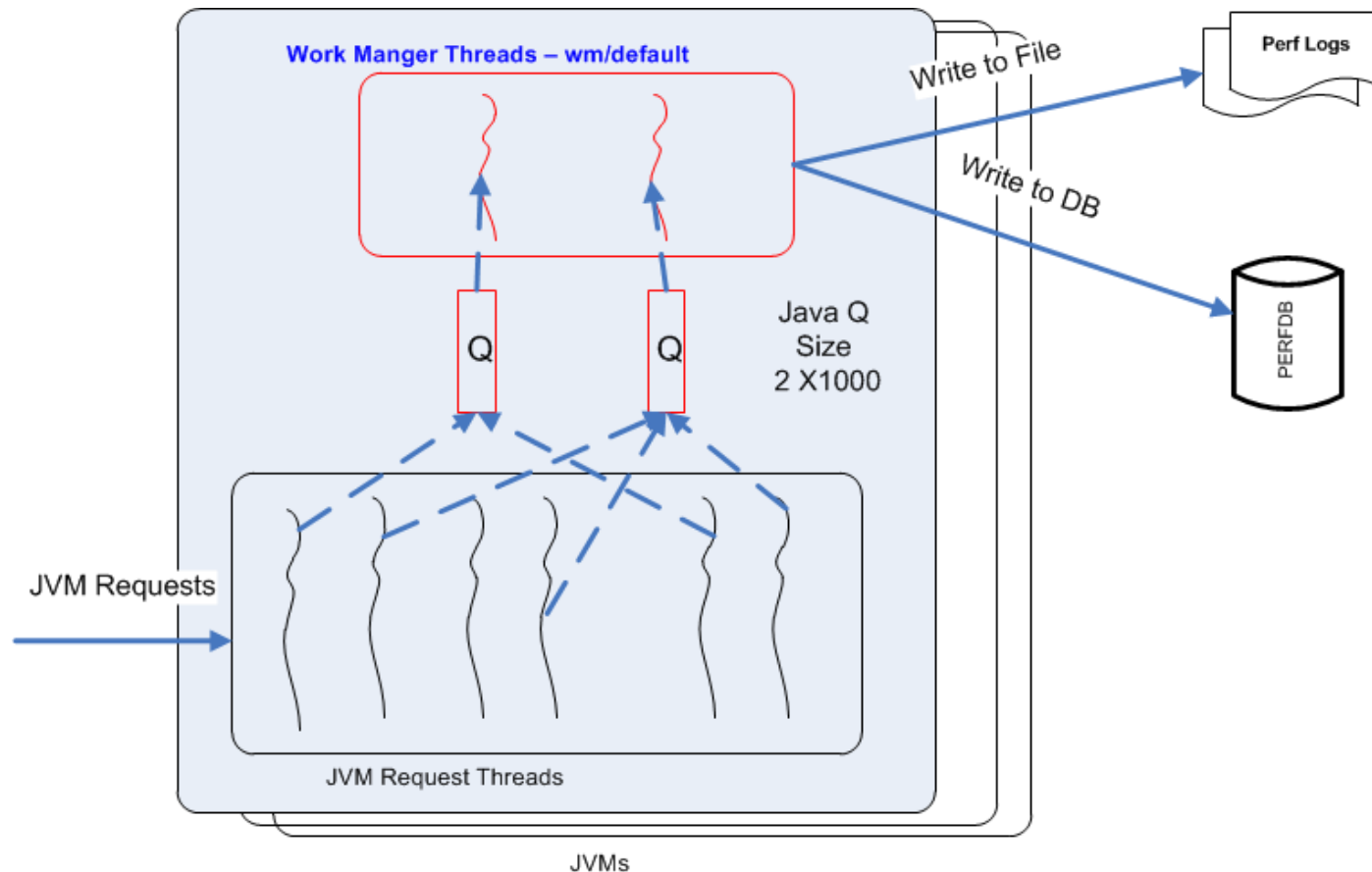
- By default PerfLog is set up to log performance metrics to System.out using `org.perf.log.logger.PerfLoggerImplStdOut` implementation
- If you are running in a J2EE server environment you can choose one of the following Asynchronous thread based implementation:
 - `org.perf.log.logger.PerfLoggerImplCommonJAsyncThread`
 - for any J2EE server that supports Common J Work Manager **(Preferred)**
 - `org.perf.log.logger.PerfLoggerImplWasWmAsyncThread`
 - for any IBM WebSphere Environment that does not support Common J work manager.
 - All IBM WebSphere Application Server version 7.0 and later supports Common J Work manager
- `org.perf.log.logger.PerfLoggerImplJMSQ` uses JMS/Q based logging
 - Useful when you want to centralize logging and have additional mediation/transformation needs on the PerfLog Data
- High level deployment diagram for both Asynchronous thread and JMS/Q based implementation is provided in the next two slides.
 - Both these implementation writes to `perf-logXX.log` log file and also to performance database
 - Logging to log files and performance database can be enabled or disabled independently.

PerfLog – Deployment – Asynchronous Thread Implementation (Preferred Logger Implementation)



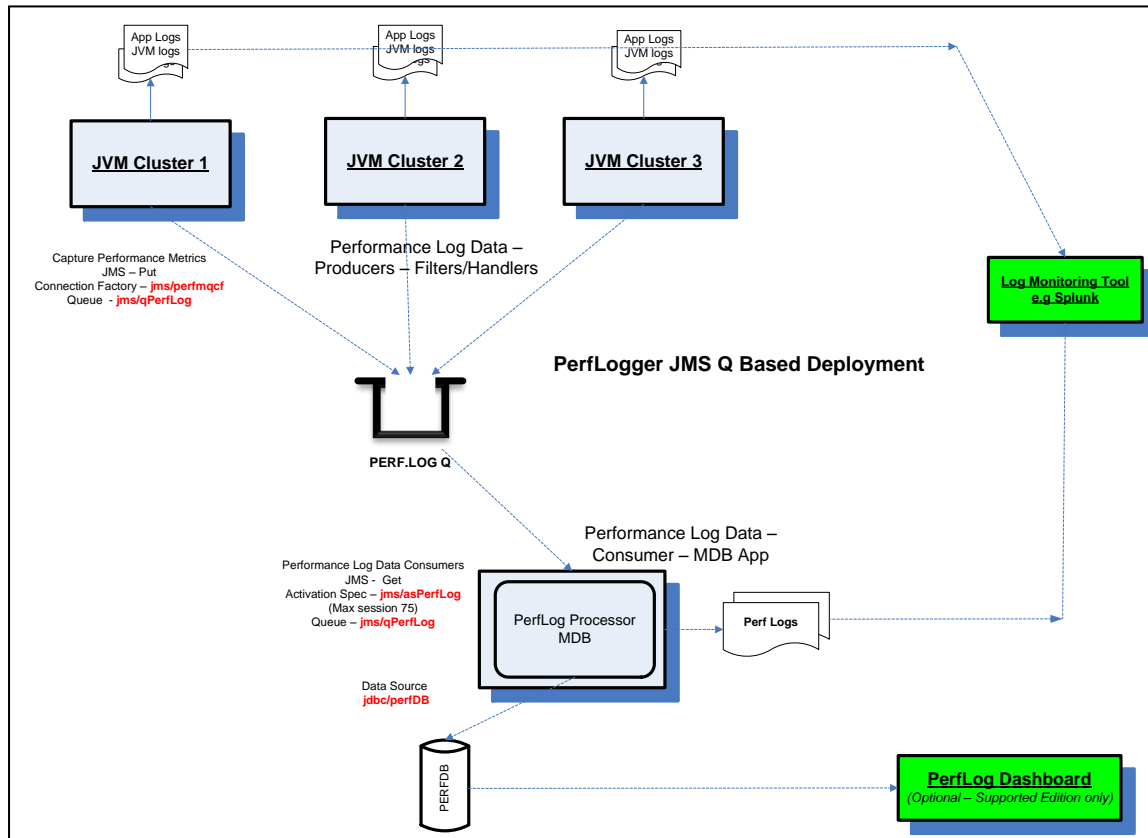
- Override properties to select the desired PerfLogger implementation by including perfLog.properties in the application or via application classpath
- Refer sample perfLog.properties file for relevant properties

PerfLog Internal – Asynchronous Thread based implementation overview



PerfLog – Asynchronous Thread Logger Implementation

PerfLog – Deployment – JMS/Q Based Logging



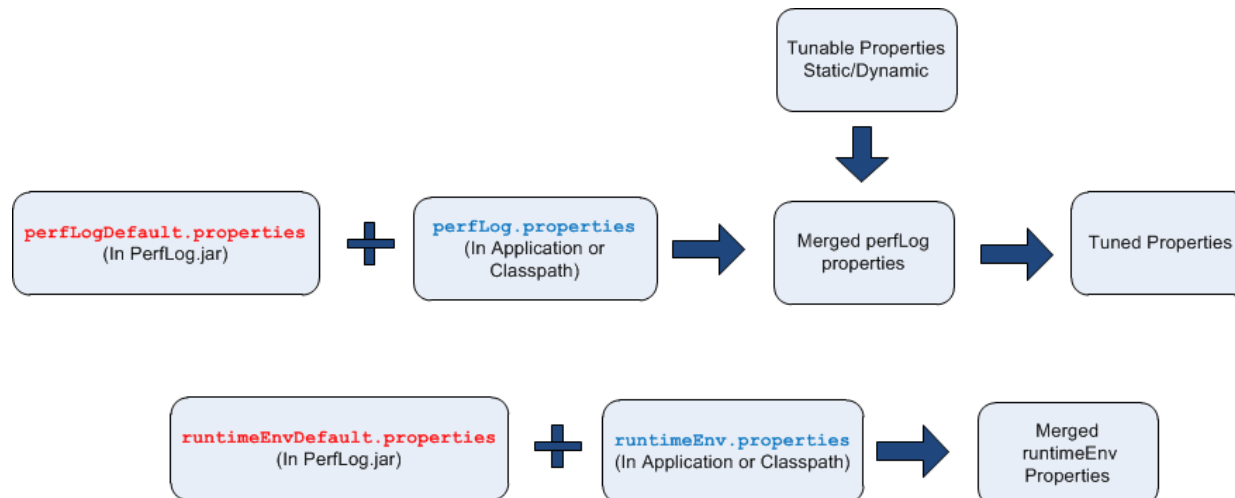
- Override properties to select the desired PerfLogger implementation by including perfLog.properties in the application or via application classpath
- Refer sample perfLog.properties for more information on relevant properties
- High scalability may require multiple queues and deployment of MDB into a clustered environment

PerfLog – Property Files



- PerfLog.jar includes two default property files
 - **perfLogDefault.properties**
 - all properties related with PerfLog to select logger implementation, cache sizes, enable/disable flags etc.
 - **runtimeEnvDefault.properties**
 - Properties related with your specific J2EE environment.
- Application can **override** the default properties by included the following properties
 - **perfLog.properties** overrides properties defined in **perfLogDefault.properties**
 - **runtimeEnv.properties** overrides properties defined in **runtimeEnvDefault.properties**
- Overriding properties file can be included in the src folder of the application source files or placed in the application class path
- Overriding property files need not include all the properties. The active properties for PerfLog is the merged properties set as shown:
 - **perfLogDefault.properties** (in PerfLog.jar) + **perfLog.properties** (in application or application classpath)
 - **runtimeEnvDefault.properties** (in PerfLog.jar) + **runtimeEnv.properties** (in application or application classpath)

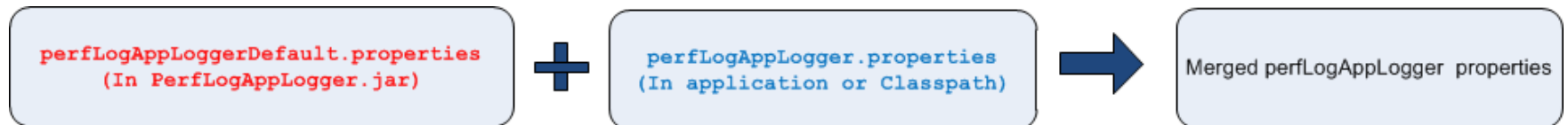
txnThresholdOverride.properties is used to override the response time threshold for specific transaction names. There is no default for this. It is only included in the application or application class path. Refer to sample properties for more info.





PerfLogAppLogger – Property Files

- PerfLogAppLogger.jar default property files
 - **perfLogAppLoggerDefault.properties**
 - Defines all properties related with PerfLogAppLogger such as logfile name, size, rotation and whether to use asynchronous thread based logging etc.
- Application can **override** the default properties by included the following properties
 - **perfLogAppLogger.properties** overrides properties defined in **perfLogAppLoggerDefault.properties**
- Overriding property – **perfLogAppLogger.properties** file can be included in the src folder of the application source files or placed in the application class path
- Overriding property files need not include all the properties. The active properties for PerfLogAppLogger is the merged properties set as shown:
 - **perfLogAppLoggerDefault.properties** (in PerfLogAppLogger.jar) + **perfLogAppLogger.properties** (in application or application classpath)





Tunable Properties Implementation

- PerfLog includes a tunable properties feature, where properties can be additionally overridden by a tunable properties implementation
- Certain properties for PerfLog are marked dynamic. These properties are read at regular intervals.
 - Properties that are dynamic are pre-fixed with `dynamic.*`
- These properties are useful when properties have to be changed at runtime without re-cycling the JVM or re-deploying the application.
- By default Tunable Properties are disabled.
 - See `perfLog.properties` for property - `static.logger.tunableProperties.enabled`
- PerfLog comes with two different implementation for Tunable Properties
 - `org.perf.log.properties.NSBTunablePropertiesImpl` – Name Space Binding tunable properties – works in IBM WebSphere environment
 - `org.perf.log.properties.URLResourceTunablePropertiesImpl` – URL based tunable properties where property file can be accessed via a Java URL path.
 - Each implementation have additional properties that define their behavior. See sample `perfLog.properties` for more details
 - Additional implementation can be easily provided by providing an implementation of `org.perf.log.properties.TunableProperties` interface and specifying the desired implementation in `perfLog.properties`.



Key PerfLog properties – Property File – perfLog.properties

- **static.logger.perfLoggerImplClass**
 - Specify PerfLogger implementation Class. Choice values are given below
 - **Default:** `org.perf.log.logger.PerfLoggerImplStdOut`
 - Writes performance log to Standard output
 - `org.perf.log.logger.PerfLoggerImplCommonJAsyncThread`
 - Preferred – Needs J2EE server with Common J Work Manager support (post J2EE 1.4)
 - Works with IBM WebSphere Application Server 6.1+
 - `org.perf.log.logger.PerfLoggerImplWasWmAsyncThread`
 - Works in all IBM WebSphere Application server environment
 - `org.perf.log.logger.PerfLoggerImplJMSQ`
 - JMS/Q based logging
 - Requires Message Drive Bean application to de-queue and write to file and database
 - There are additional properties that go with each of the above implementation. Refer to perfLog.properties for more information.
- **dynamic.logger.perfLoggerImpl.logEnabled**
 - **Default** : true
 - Enables / Disables performance Logging
- **dynamic.perflog.context.responseTimeThresholdInMillis**
 - **Default** : 30,000 i.e. 30 seconds
 - Property to set the global response time threshold. When response time threshold is exceeded, PerfLog dumps the PerfLogContext data to System.out that could aid in diagnosis of performance issues.
 - Set depending on your application needs
 - Use `txnThresholdOverride.properties` file in your application or application classpath to override transaction names with specific response time threshold values

For other properties refer documentation in sample perfLog.properties



Key PerfLog properties – Property File – runtimeEnv.properties

- **runtime.env.containerType** - Indicates the JVM container type.
 - **Default** : default
 - Possible values are:
 - websphere, tomcat, oracle, weblogic, jboss, glassfish, standalone
- **runtime.env.JvmCloneGetterImpl** - returns unique name for the JVM in a clustered environment
 - **Default** : `org.perf.log.utils.DefaultJvmCloneGetterImpl`
 - This implementation class is used to get the unique name of the JVM. The unique name of the JVM in a clustered environment is vendor dependent. Tomcat and WebSphere implementation class are provided. Other implementations exist but currently defaults to the default implementation. The default implementation returns the PID@hostname for the JVM. The unique name of the JVM is also prefixed to the GUID to identify which JVM created the original GUID that is used to track requests if the request is tracked across multiple JVMs.
 - Set to one of the following based on your environment
 - `org.perf.log.utils.WebSphereJvmCloneGetterImpl` for WebSphere
 - `org.perf.log.utils.TomcatJvmCloneGetterImpl` for Tomcat
- **runtime.env.PortletInfoGetterImpl** - Implementation returns the portlet name and page name where portlet is invoked.
 - **Default** : `org.perf.log.utils.DefaultPortletInfoGetterImpl`
 - Possible values are:
 - `org.perf.log.utils.WebSpherePortletInfoGetterImpl` – Implementation for IBM WebSphere Portal.
 - `org.perf.log.utils.LiferayPortletInfoGetterImpl` - Implementation for Liferay Portal container. Currently returns the default.



PerfLogAppLogger properties – Property File – perfLogAppLogger.properties

```
#Values for logDestination: console, file
#if logDestination is file, then additional
#properties for logFile* will be used
logDestination=console
#logDestination=file
logFileRootDir = /tmp
logFileMaxSize = 2097152
logFileNumToKeep=10
#Change the file name for your application
logFileName=appLogFile
#Valid values : info, debug, warn, error, trace, off
logFileInitialLevel=info
#
#Use AsynchronousLogging - true or false
#Default is false
#Set to true if you run in a J2EE container
#and require asynchronous logging
#The Asynchronous Logging is implemented
#using commonJ work manager thread
#
useAsynchronousLogging=false
#Number of asynchronous logger threads to created
commonJAsyncThreadLogger.numAsyncLoggerTaskThreads=2
#Default buffer for each thread
commonJAsyncThreadLogger.maxQSize=1000
#Default work manager thread pool name
#For WebSphere Environment wm/default is always present
#For other J2EE environment check Vendor manual or create
#a thread pool as per Vendor documentation and specify the
#name for this property
commonJAsyncThreadLogger.workManagerTheadPoolResourceName=wm/default
#Asynchronous thread sleep time when idle
commonJAsyncThreadLogger.threadSleepTimeInMillis=10000
#print asynchronous thread logging statistics every 10 minutes
commonJAsyncThreadLogger.printStateTimeIntervalInMillis=600000
```

This is the property file for PerfLogAppLogger. If application runs in a J2EE Server, there is a choice to log asynchronously using Common J Work Manager thread.

Pexus PerfLog - Editions



Features	Community Edition (CE)	Supported Edition (SE) – for IBM WebSphere Environment	Customization Service
PerfLog Source code	X http://github.com/pexus http://git.pexus.net/perflog	Available on request	<p>Pexus LLC Services are available to customize and integrate Pexus PerfLog with any J2EE environment and J2EE application Call: 1-888-950-4442 Or send e-mail to sales@pexus.com</p>
Binaries – PerfLog.jar, PerfLogAppLogger.jar	X	X	
Sample Binaries		X	
Support – e-mail and phone support		X	
PerfLog Dashboard – Visualization software for Performance Data with real time alerting		X	
Updates	X	X	
Pricing	Free Apache 2.0 License	Priced/JVM Send e-mail to sales@pexus.com	

Resources

- PerfLog Community Edition Sources
 - <https://github.com/pexus/PerfLog>
 - <http://git.pexus.net/perflog>
- PerfLog Binary Downloads and Javadocs
 - <http://www.pexus.com/perflog>
- J2EE APIs
 - <http://docs.oracle.com/javaee/>
- PerfLog Supported Edition for IBM WebSphere and Customization Services
 - <http://www.pexus.com/perflog>
 - E-mail : sales@pexus.com

