CHAPTER 1

Using the Log Service

Overview of the JES Log Service

When you start the JES log service, it runs along with the JES framework, in the background, waiting for event messages. The log service runs when you start the JES framework and then enter a command like this one:

> **start log**, servlet, http, tcatjspcruntime, httpauth, httpusers, jesmp

An event message can come from any type of event, including a BundleEvent, FrameworkEvent, or ServiceEvent, or from any bundle. The log resides in memory as long as the JES framework is running and can hold up to 1,000,000 entries. However, the log is not stored persistently and is destroyed when the framework is shut down.

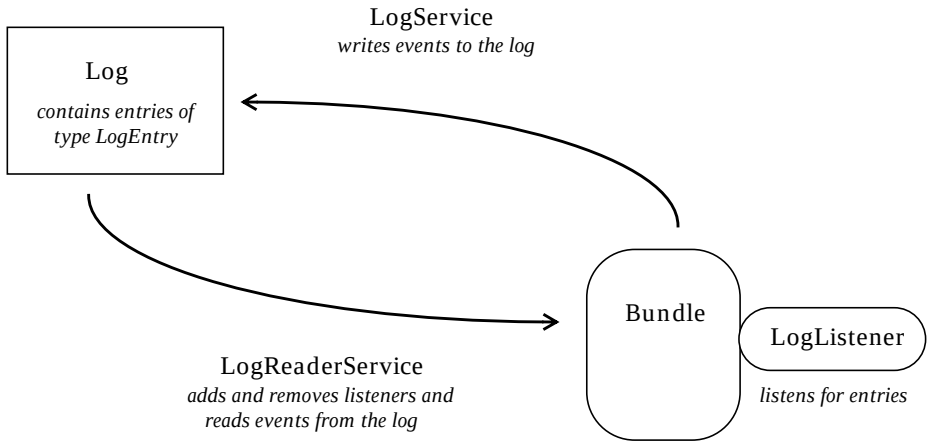
Most of the log service API is contained in the package org.osgi.service.log, which has four interfaces—LogEntry, LogListener, LogReaderService, and LogService.

You can think of the interfaces in the log service API this way:

- LogEntry creates an entry to add to the log.
- LogService puts the entry into the log.
- LogListener creates a listener so that a bundle is notified when an entry is added to the log.
- LogReaderService adds and removes listeners and reads messages stored in the log.

If you were to draw the log service, it might look something like FIGURE 1-1.

FIGURE 1-1 The Interfaces in the Log Service API



The JES framework also has a configurable interface with features that add to the OSGi specification; that is discussed later, in “Discovering the Configurable Interface.”

Adding Entries to the Log

Each log entry is a a message that describes an event. A message always includes a log level and a String, and can also include a Throwable exception, a ServiceReference object, or both. You log messages using one of these forms of the log method:

LogService

public void **log**(int level, String message)
public void **log**(int level, String message, Throwable exception)
public void **log**(ServiceReference sr, int level, String message)
public void **log**(ServiceReference sr, int level, String message, Throwable exception)

Each entry contains a log level. The log levels are also in LogService and range from 1 to 4, with these descriptions:

Log Level	Description	Meaning
1	LOG_ERROR	An error message indicating that the bundle or service may not be functional. Highest level.
2	LOG_WARNING	A warning message indicating that the bundle or service is still functioning but may experience problems in the future as a result of the warning condition.
3	LOG_INFO	An informational message added as a result of some change in the bundle or service that does not indicate a problem.
4	LOG_DEBUG	A debugging message that is used to describe a problem and may be meaningless to anyone but the developer. Lowest level.

In this list, the highest (or most severe) log level is 1 (LOG_ERROR), and the lowest (or least severe) is 4 (LOG_DEBUG).

When you add a Throwable exception to a log entry, it's usually to an entry of type 1, LOG_ERROR. When you give an entry a ServiceReference object, the entry allows other bundles to discover information about the bundle that logged the message.

In addition to the log level, log string, Throwable exception, and ServiceReference object that you add, the log service also timestamps log entries. If all of this is sounding like a very long String for one entry, you're right. A log entry, which is a single String object, can contain multiple lines.

▼ To Log Entries

1. **Get the bundle context from the framework.**
2. **Get a reference to the log service, then the service itself.**
3. **Use the** log **method to put messages into the log.**

CODE EXAMPLE 1-1 demonstrates how to add messages to the log.

CODE EXAMPLE 1-1 Logging Messages (*LogExample.java*)

	package logexample;
Import the LogService interface	import org.osgi.framework.BundleContext; import org.osgi.framework.ServiceReference; import org.osgi.service.log.LogService;
	public class LogExample {
Get the bundle context from the framework	LogExample(BundleContext bc) { try {
Use the bundle context to create the log service reference	getServices(bc);

log is defined in getServices, below
 	
Check for a null in case getServices is called repeatedly
	if (log == null) {
	System.err.println("Unable to get the Log service reference!");
	return;
	}

Generate some fake log messages	log.log(LogService.LOG_ERROR, "Log error");
	log.log(LogService.LOG_WARNING, "Log warning");
	log.log(LogService.LOG_INFO, "Log info");
	log.log(LogService.LOG_DEBUG, "Log msg");

Set log to null to avoid memory leaks	log = null;
	
	} catch (Exception e) {
	System.err.println(e.getMessage());
	}

	}
getServices is defined here	public final static String LOG_CLASS = "org.osgi.service.log.LogService";

A reference to the log service object	private LogService log = null;
Synchronized to prevent the log service from being uninstalled while this bundle is trying to get it	synchronized void getServices(BundleContext bc) { ServiceReference sr = null;

Get the service reference and the log service	if (log == null) {
	if ((sr = bc.getServiceReference(LOG_CLASS)) != null) {
	log = (LogService) bc.getService(sr);
	}
	}
	}
	
	}

Getting Information from a Log Entry

Because each entry in the log is a LogEntry object, you can use the LogEntry methods to get information from the entry. You can extract the log level, log string, Throwable exception, ServiceReference object, or timestamp that are part of the entry. You can also retrieve the bundle that logged the message. The timestamp is returned in the number of milliseconds since January 1, 1970, 00:00:00 GMT.

The methods you use to retrieve information from a LogEntry object are these:

LogEntry

public Bundle **getBundle**()
public ServiceReference **getServiceReference**()
public int **getLevel**()
public String **getMessage**()
public Throwable **getException**()
public long **getTime**()

But you may also want to read the entries that were already added to the log before you created the listener. To do this, you enumerate through the entries and read them.

▼ To Enumerate and Read the Entries in the Log

1. **Get references to the log and log reader services.**
2. **Get the services themselves.**
3. **Use the** getLog **method from the log reader service to get the entries in the log as an enumeration.**
4. **Move through each entry in the enumeration.**
5. **Use the methods from LogEntry to read the entries.**

CODE EXAMPLE 1-2 shows you how to enumerate through the log.

CODE EXAMPLE 1-2 Enumerating Through the Log (*LogEnumeration.java*)

	package logenumeration; import org.osgi.framework.Bundle; import org.osgi.framework.BundleContext; import org.osgi.framework.ServiceReference;
Import the log classes	import org.osgi.service.log.LogEntry; import org.osgi.service.log.LogService; import org.osgi.service.log.LogReaderService; import java.util.Date; import java.util.Enumeration;

	public class LogEnumeration {
	
	private static final String API_PREFIX = "API Test ";
	
	LogEnumeration(BundleContext bc) {
	try {

Get references to the LogService and LogReaderService	getServices(bc);

Display a message if a service is null	if (log == null) {
	System.err.println("Unable to get Log service reference!");
	return;
	}
	
	if (logreader == null) {
	System.err.println("Unable to get LogReader service reference!");
	return;
	}

	// API test #1 System.out.println("public void log(int level, String msg)");

Log entries with just a level and a message	log.log(LogService.LOG_ERROR, API_PREFIX + "1: Log error");
	log.log(LogService.LOG_WARNING, API_PREFIX + "1: Log warning");
	log.log(LogService.LOG_INFO, API_PREFIX + "1: Log info");
	log.log(LogService.LOG_DEBUG, API_PREFIX + "1: Log debug");

Log entries with a level, message, and Throwable exception

Log entries with a level, message, and service reference object

Log entries with a level, message, service reference object, and Throwable exception

Enumerate through the log and show each entry's contents

```
        // API test #2

        System.out.println( "public void log(int level, String msg, Throwable e)" );

        log.log( LogService.LOG_ERROR,   API_PREFIX + "2: Log error",
            new IllegalArgumentException("error") );
        log.log( LogService.LOG_WARNING, API_PREFIX + "2: Log warning",
            new IllegalArgumentException("warning") );
        log.log( LogService.LOG_INFO,    API_PREFIX + "2: Log info",
            new IllegalArgumentException("info") );
        log.log( LogService.LOG_DEBUG,   API_PREFIX + "2: Log debug",
            new IllegalArgumentException("debug") );


        // API test #3

        System.out.println( "public void log(ServiceReference sr, int level, String msg)" );

        log.log( sr, LogService.LOG_ERROR,   API_PREFIX + "3: Log error" );
        log.log( sr, LogService.LOG_WARNING, API_PREFIX + "3: Log warning" );
        log.log( sr, LogService.LOG_INFO,    API_PREFIX + "3: Log info" );
        log.log( sr, LogService.LOG_DEBUG,   API_PREFIX + "3: Log debug" );


        // API test #4

        System.out.println( "public void log(ServiceReference sr, int level, String msg,
            Throwable e)" );

        log.log( sr, LogService.LOG_ERROR,   API_PREFIX + "4: Log error",
            new IllegalArgumentException("error") );
        log.log( sr, LogService.LOG_WARNING, API_PREFIX + "4: Log warning",
            new IllegalArgumentException("warning") );
        log.log( sr, LogService.LOG_INFO,    API_PREFIX + "4: Log info",
            new IllegalArgumentException("info") );
        log.log( sr, LogService.LOG_DEBUG,   API_PREFIX + "4: Log debug",
            new IllegalArgumentException("debug") );


        System.out.println();

        for( Enumeration e = logreader.getLog(); e.hasMoreElements(); ) {
            showEntry ( (LogEntry) e.nextElement() );
        }

    } catch (Exception e) {
        System.err.println( e.getMessage() );
    }

}
```

	private final static String LOG_CLASS = "org.osgi.service.log.LogService";
	private final static String LOG_READER_CLASS = "org.osgi.service.log.LogReaderService";
	
	private LogService log = null;
	private LogReaderService logreader = null;
	
	ServiceReference sr = null;
	
	synchronized void getServices(BundleContext bc) {
	if (log == null) {

Reference to the LogService object	if ((sr = bc.getServiceReference(LOG_CLASS)) != null) {
	log = (LogService) bc.getService(sr);
	}

Reference to the LogReaderService object	if ((sr = bc.getServiceReference(LOG_READER_CLASS)) != null) {
	logreader = (LogReaderService) bc.getService(sr);
	}
	}
	}

Show the contents of a log entry	private boolean showEntry(LogEntry entry) {
	
	Bundle b = entry.getBundle();
	int level = (int) entry.getLevel();
	long time = entry.getTime();
	String message = entry.getMessage();
	ServiceReference sr = entry.getServiceReference();
	Throwable e = entry.getException();

Returns true if there's an error in the entry	
Show only those entries that were logged by this bundle	if (message.startsWith(API_PREFIX) == true) {
Show the severity level information	System.out.print(" Level " + level + ": "); switch (level) { case LogService.LOG_ERROR: System.out.print("ERROR "); break;

	case LogService.LOG_WARNING: System.out.print("WARNING"); break; case LogService.LOG_INFO: System.out.print("INFO "); break; case LogService.LOG_DEBUG: System.out.print("DEBUG "); break; default: System.out.print(" ??? "); return true; }
Show the time that the message was logged	System.out.println(" at " + new Date(time).toString());
Show the message that was logged	System.out.println(" Message: " + message);

Show the service reference information and any exception messages	System.out.print("Service, Exception: " + sr);
	if (e == null) {
	System.out.println(", None.");
	} else {
	System.out.println(", '" + e.getMessage() + "'");
	}
	System.out.println();
	} // if
	return false;
	}
	}

Creating a Listener

To create a listener for your bundle, you implement both the LogListener and LogReaderService interfaces. LogListener creates the listener that allows your bundle to "hear" entries logged by the JES framework or by other bundles as soon as they have been logged. LogReaderService allows you to add a listener, remove it, or read the entries already in the log in the form of an Enumeration.

LogListener and LogReaderService include these methods:

LogListener

public void **logged**(LogEntry entry)

LogReaderService

public void **addLogListener**(LogListener listener)
public void **removeLogListener**(LogListener listener)
public Enumeration **getLog**()

▼ To Create a Listener

1. **Write a class that implements LogListener.**
2. **Use the bundle context to create a reference to the log and log reader services.**
3. **Use the references to get the services themselves.**
4. **Add the log listener to the bundle.**
5. **Make the bundle sleep briefly so that framework events can be heard first.**
6. **Log the messages.**
7. **Remove the log listener at the end of the class, so that it is removed when the bundle shuts down.**

CODE EXAMPLE 1-3 shows you how to create a log listener.

CODE EXAMPLE 1-3 Creating a Log Listener (*LogListening.java)*

	package loglistening; import org.osgi.framework.BundleContext; import org.osgi.framework.ServiceReference;
Import the log service classes	import org.osgi.service.log.LogEntry; import org.osgi.service.log.LogListener; import org.osgi.service.log.LogService; import org.osgi.service.log.LogReaderService;
Implement LogListener	public class LogListening implements LogListener {

	LogListening(BundleContext bc) {
	try {
	getServices(bc);

Use the bundle context to create the log service reference	if (log == null) {
	System.err.println("Unable to get Log service reference!");
	return;
	}
	if (logreader == null) {
	System.err.println("Unable to get LogReader service reference!");
	return;
	}

Set up a listener	logreader.addLogListener(this);
Make the bundle sleep briefly so framework events can be heard first	Thread.sleep(10);

Create some messages to be logged	System.out.println("Using: public void log(int level, String msg)");
	log.log(LogService.LOG_ERROR, "Sample LOG_ERROR");
	log.log(LogService.LOG_WARNING, "Sample LOG_WARNING");
	log.log(LogService.LOG_INFO, "Sample LOG_INFO");
	log.log(LogService.LOG_DEBUG, "Sample LOG_DEBUG");

Remove the listener so we can't detect messages any more	logreader.removeLogListener(this);
Messages that follow should not be logged	log.log(LogService.LOG_ERROR, "ERROR: You shouldn't see this");

Remove the log listener when the bundle shuts down	if (log == null) {
	if ((sr = bc.getServiceReference(LOG_CLASS)) != null) {
	log = (LogService) bc.getService(sr);
	}
	if ((sr = bc.getServiceReference(LOG_READER_CLASS)) != null) {
	logreader = (LogReaderService) bc.getService(sr);
	}
	}
	}

	private final static String LOG_CLASS = "org.osgi.service.log.LogService"; private final static String LOG_READER_CLASS = "org.osgi.service.log.LogReaderService";
Reference to the LogService object	private LogService log = null;
Reference to LogReaderService	private LogReaderService logreader = null;
Here's getServices	synchronized void getServices(BundleContext bc) { ServiceReference sr = null;

Implement the logged method from LogListener	public void logged(LogEntry entry) {
	try {
	System.out.println(entry.getMessage());
	} catch (Exception e) {
	System.err.println(e.getMessage());
	return;

Gets messages as they are logged and displays them	} }
	}

Discovering the Configurable Interface

The JES log service includes methods you can use to configure the size and severity threshold of the log. Please note that these methods are not made publicly available in the JES API. You must discover them by using reflection with a configuration object (an object that implements org.osgi.framework.Configurable).

The methods that you use to configure the log are listed below:

public int **getLogSize**()
public synchronized **setLogSize**(int size)
public int **getSeverityThreshold**()
public void **setSeverityThreshold**(int threshold)

When you use reflection to discover these methods, the output of the reflection looks like this:

```
Method: getLogSize
Current Log size = 20 LogEntry elements.

Method: setLogSize
Size should be set to 50

Method: getLogSize
New Log size = 50 LogEntry elements.

Method: getSeverityThreshold
Current severity threshold = 4

Method: setSeverityThreshold
Severity threshold should be set to 2

Method: getSeverityThreshold
New severity threshold = 2
```

When you configure the log's size, remember that its minimum size of is 1, its maximum size is 1,000,000, and its default size is 20. If you set the size of the log less than the current size, only the most recently logged messages are kept. If you make the size of the log greater than the current size, the log service creates a new log, copies all previous messages to it, and adds space for the additional entries.

The severity threshold is the log level at or below which entries are added to the log. For example, if you set the severity level to 4, entries with a log level of 4 or lower (that is, all entries) are added to the log. The severity threshold can be between 1 and 4, unless you extend the LogService interface to add more levels.

When you set the severity threshold to a new value, future log entries are only placed in the log if their levels are at or below the severity threshold level. When the threshold level is 4 (LOG_DEBUG, the default), entries of all levels are added to the log.

The log size and severity threshold are stored in a properties file on the embedded server's filesystem (if one exists) after the JES framework is shut down.

You can find an example of how to use reflection to discover the configurable interface in your jes2.0/docs/examples/LogReflect.

Using the Log Properties

You can also set the log size and severity threshold when you start the JES framework. Start JES 2.0 from the command line, using a java command with the -D option, followed by a property name and value, for example:

```
java -Dcom.sun.jes.service.log.size=200000 -jar framework.jar
```

The properties that affect the log service and their values are listed below:

com.sun.jes.service.log.size	Minimum 1, maximum 1000000, default 20
com.sun.jes.service.log.threshold	1, 2, 3, or 4; default 4