Logger Configuration

Goals

After completing this section, students should be able to do the following:

- **Configure the categories of loggers.**
- **Configure the root logger.**

define loggers

Java developers use logging extensively in their source code as a means of communicating application state based on certain events.
Not all log events should be considered as a sign of a problem with the application: a log event can be used, for example, to communicate correct operation. In a working EAP server, each individual log event that appears within the Java code can be important or not important.
Through the use of loggers, such events can be filtered in the following ways:

- **specifying the logging level, and**
- **specifying a category name.**

The category (category) logger defines the source of the event. It is standard convention to use a fully qualified Java class name as the registration category in Java source code, but the category name can be any string that makes sense to the developer.

The logger level defines the severity of the event. For example, it represents a fatal error from which the application cannot recover, or it only provides debugging information to the developer.

A logger is defined using the <logger> tag within the logging subsystem configuration and the format is as follows:

```
<logger category="java_class_and/or_package_name">
        <level name="log_level"/> </
logger>
```

The logger categories follow a hierarchical rule. If a category is defined for a given package, all classes in that package and all subpackages are included in the category. For example, the following logger sets the logging level to WARN for any Java class in the org.jboss.jca package:

```
<logger category="org.jboss.jca"> <level
        name="WARN"/> </logger>
```

The CLI command to define this logger is as follows:

```
/profile=default/subsystem=logging/logger=org.jboss.jca:add(\
        category=org.jboss.jca,level=WARN)
```

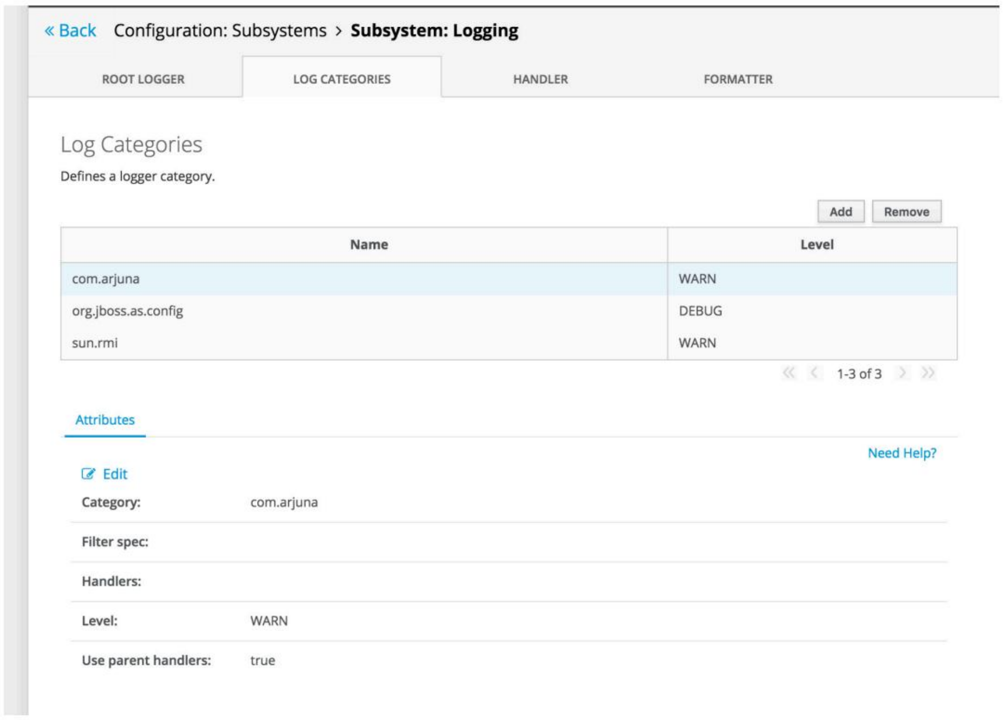
The category can be as specific as you like, including configuring a logger for a single class. For example, the following logger sets the logging level to DEBUG only for the org.jboss.jca.core.naming.JndiBinder class:

```
<logger category="org.jboss.jca.core.naming.JndiBinder"> <level
        name="DEBUG"/> </logger>
```

The CLI command to define this logger is as follows:

```
/profile=default/subsystem=logging/logger=org.jboss.jca.core.naming.JndiBinder:\
        add(category=org.jboss.jca.core.naming.JndiBinder,level=DEBUG)
```

The loggers can be configured and managed using the EAP administration console in the Configuration > Subsystems > Logging section. Click Log and then the Log Categories tab to view, edit, or add loggers identified by their category attributes:



Log Categories

Defines a logger category.

Name	Level
com.arjuna	WARN
org.jboss.as.config	DEBUG
sun.rmi	WARN

Category:	com.arjuna
Filter spec:	
Handlers:	
Level:	WARN
Use parent handlers:	true

Figure 7.11: Log Categories

A logger can also specify which handler to use. Consider the following example:

```
<logger category="org.jboss.as" >          1
        <level name="INFO" />  2
        <handlers>
                <handler name="FILE" />  3
        </handlers>
```

```
</logger>
```

1. The category attribute represents the name of the logger and assigns a fully qualified class or package name. For example, if the application is in the com.mycompany.myapp package, a new logger can be defined by adding <logger category="com.mycompany.myapp">.
2. The level attribute sets the default logging level for this logger. The logger accepts all logging events with the same or higher priority.
3. The handler attribute denotes a valid handler reference defined in the configuration file. Multiple handlers can be listed within the handlers tag. The logger sends all relevant messages to the defined handlers, as well as to handlers inherited from ancestor loggers.

Understanding the hierarchy of loggers

Loggers are arranged in a tree-like hierarchy, based on their category value. The period (.) is used as a separator between hierarchy levels, following the Java package naming rules. If a logger is defined for a subpackage of another logger, it becomes a child of that logger. For example, com.mycompany.onlinestore is a child logger of com.mycompany.

EAP applications and components always instantiate at runtime the specific secondary logger they require. Following the example above, an application Java class named com.mycompany.onlinestore.CheckOut creates a logger using its name as the category.

If a logger is NOT configured explicitly, the configuration of its parent logger is used. If the main logger is not configured either, the configuration of the main logger of the main logger is used, and so on. The root logger serves as the default configuration for all loggers that were not explicitly configured, as well as any of its ancestors.

Generally, a specific logger is only configured when it (or its child loggers) requires a different level value or a different handler. Otherwise, the values of an ancestor will be used.

The following example shows a configuration in which events from the log of Generic EAP filters at a very high level (WARN), but the JBeret subsystem gets a lower level (INFO), so it has more events sent to any handler inherited from the root logger:

```
<logger category="org.wildfly">
        <level name="WARN"/>
</logger>
<logger category="org.wildfly.jberet"> <level
        name="INFO"/> </logger>
```

The logger configurations inherit, as in the previous example, all the root logger handlers. Most administrators assume that it would be enough to configure a different handler on any logger so that its events are sent ONLY to that handler, although this is NOT true: a logger always inherits all parent handlers, UNLESS this behavior is disabled by changing the use-parent-handlers attribute of the logger.

For example, the following example sends all online store application events to the handler named STORE_APP_HANDLER, as well as any handlers configured by the root logger:

```
<logger category="com.mycompany.onlinestore">
        <level name="INFO"/>
        <handlers>
                <handler name="STORE_APP_HANDLER"/>
        </handlers> </
logger>
```

That is, log events generated by application classes can be sent to multiple handlers, even if the logger configuration names only one handler.

If the intended result is that all application log events are sent only to the STORE_APP_HANDLER and not to the handlers configured by the root logger, set the use-parent-handlers attribute to false:

```
<logger category="com.mycompany.onlinestore" use-parent-handlers="false"> <level name="INFO"/
        > <handlers>

                <handler name="STORE_APP_HANDLER"/>
        </handlers> </
logger>
```

Root Logger Configuration

Loggers have a hierarchy according to which the child loggers inherit the values of the parent loggers. At the top of the logger hierarchy, there is a single logger called *the root logger* . The root logger is a kind of universal logger in the sense that its values are associated with each individual logging event that is not specifically associated with a child logger. Because they reside at the top of the logger hierarchy, the root logger's values are inherited by all other loggers, and all other loggers can override these values.

In the EAP server configuration files, the root logger is configured as follows manner:

```
<root-logger>
        <level name="INFO"/>
        <handlers>
                <handler name="CONSOLE"/>
                <handler name="FILE"/>
        </handlers> </
root-logger>
```

Note that the default logging level for all events is INFO, and these events are logged to the console and with the FILE handler, which is the server.log file.

The root logger can be configured according to a specific need. For example, if on the production server, logging is at the ERROR level and you need to disable console logging, the following configuration will meet this requirement:

```
<root-logger>
        <level name="ERROR"/>
        <handlers>
```

```
                <handler name="FILE"/>
        </handlers> </
root-logger>
```

The logging subsystem has several CLI operations pertaining to the root logger. These are available in the /subsystem=logging/root-logger=ROOT: namespace of the CLI:

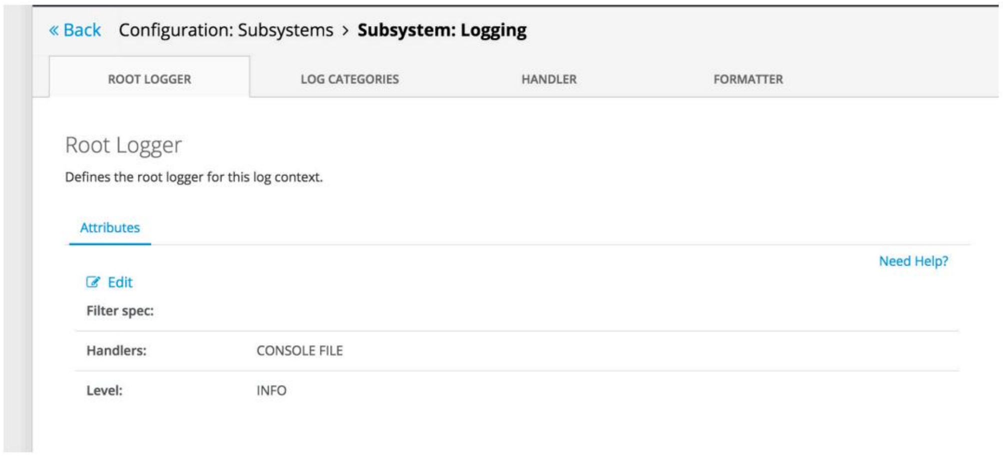
- change-root-log-level: change only the log level attribute of the logger root.
- set-root-logger – Assigns a new definition to the root logger.
- remove-root-logger – removes the entire definition of the root logger.
- root-logger-assign-handler: defines the handlers for the root logger.
- root-logger-unassign-handler: removes a handler from the list of handlers of the root logger.

For example, the following command changes the root logger level to ERROR in the default profile:

```
/profile=default/subsystem=logging/root-logger=ROOT:change-root-log-level(level=ERROR)
```

Of course, the level and handler attributes of the root logger and other logger objects can be modified directly, without having to resort to these specialized operations. These operations exist to simplify making changes, such as adding or removing a single handle.

The root logger can also be configured and managed using the EAP administration console in the Configuration > Subsystems > Logging section. Click the Log button and then the ROOT LOGGER tab to view, edit, or add root loggers:



Root Logger

Defines the root logger for this log context.

Filter spec:

Handlers: CONSOLE FILE

Level: INFO

Figure 7.12: ROOT Logger

Log Level Priority

Log levels are arranged in decreasing order of detail (lower levels in the priority order result in more detailed logs). Each log event is label with specific levels. The loggers and handlers can be set to the minimum level so as not to be dropped. From highest to lowest, the log level priorities are:

log level	Description
OFF	special level that disables all logging.
FATAL	A fatal event signals that the service cannot continue.
SEVERE	A serious event indicates a serious failure in which one or more EAP services failed.
ERROR	A problem that can interrupt the provision of the service.
WARNING	A Warning event indicates a possible problem.
WARN	A potentially harmful problem has occurred.
INFO	Information about an event. It does not indicate any type of error.
CONFIG	This is a message level for static configuration messages.
FINE	This is a message level that provides tracking information.
DEBUG	Useful additional information for debugging errors.
TRACE	Allows you to examine in depth the behavior of the JBoss server.
FINER	Indicates a more detailed trace message than FINE.
FINEST	Indicates a more detailed trace message than FINER.
ALL	Indicates that all messages should be logged.

The recommended approach on production servers is to use the ERROR or WARN level, and NOT use the INFO level and below, as much as possible to avoid detailed logs. Whenever a problem needs to be investigated, the logs can be set to a lower level for a brief period to collect and observe error messages; then rise to the ERROR or WARN level when troubleshooting is complete.

If possible, add the corresponding category loggers to the namespace of the application package or EAP subsystem being investigated, rather than changing the root logger level to avoid very detailed logging, which would cause the application to work slower due to I/O bottlenecks.

use

After changing the logging levels, it is not required to change the EAP server. Log messages at the appropriate level automatically start appearing in the log files, depending on the logger and driver settings.