

Logging

1. Overview of RADRunner Logging

RADRunner is capable of logging all actions carried out by the process engine, down to a very fine level of detail. Installation of a new system involves customising this logging, to record only those actions of interest to process auditors. For example, the completion time of each activity could be recorded, along with gives to and gets from interactions.

RADRunner can be clustered, to run on several servers at once, using the same data store. In this case, each server will write its log messages to a separate log file. Each time that a server is restarted, a new log file is also started.

Hence RADRunner produces, during the course of a process, a number of text log files, each containing a line for each action logged. The lines contain information including data and time (in milliseconds), user, role instance, Java class and method, text describing the action, etc. The format of these lines can be customised very precisely, with fields included and excluded as required, so as to permit import into an analysis tool.

For example, the default format is Comma-Separated Variable (CSV), in which each item on a line is followed by a comma. This format can be imported directly into most major spreadsheets and databases. Queries can then be made against the audit data as required.

2. Application Server Logging

The application server you use may generate its own log files, and it is generally possible to specify how these are named and where they are placed. For example, with WebSphere you can specify the name for both the standard output and standard error log files, and they are placed relative to the working directory of the application server.

For example, if you set the application server working directory to `\\myServer\rim`, you could name the standard output and error log files `log/radrunner_out.log` and `log/radrunner_error.log` to ensure that they are placed in `\\myServer\rim\log`.

3. RADRunner Log Files

RADRunner generates its own log files in addition to those created by your application

server, containing application-specific information.

In a typical corporate environment, the web and EJB modules of RADRunner may be deployed on separate machines - perhaps several machines for each module in a cluster. In this case, a strategy should be implemented for the allocation of RADRunner log files to hard disk storage.

One approach is to use a fast network-attached storage device common to all web and EJB containers. Another is to allocate one such device to the web module and another to the EJB module - this would separate the logging of page requests served by the web module from the logging of business activities carried out by the EJB module. Other strategies are also possible.

Any such strategy is implemented via 2 means:

1. The setting of the system property *com.rolemodellers.rim.logPath* (see [Application Directories](#)) in a web or EJB container controls the *logical location* where it will write RADRunner log files.
2. This logical location is mapped to a *physical device* via the drive mapping assigned to the machine running the container.

The most practical approach may be to assign a common logical location to all modules as a system property (for example, *D:\rim\log*), then map the specified drive (in this example, D:) to a specified physical device on each machine running a web or EJB container. This approach enables the application server configuration to be cloned across all containers without change, while allowing the log file mapping for each container to be changed easily as necessary. It may be necessary to do this if, for example, a particular storage device goes down.

4. RADRunner Log Output

RADRunner logging can be customised in 2 ways:

- [Logging Levels and Categories](#)
- [Log Message Format](#)

4.1. Logging Levels and Categories

RADRunner logging is implemented using the open source framework [log4j](#). If you wish to change the default settings to customize log output, this can be done in one of 2 places.

If the application server also uses log4j (as JBoss does, for instance), you will need to update the log4j configuration file used by the application server itself. This file is likely to be called either *log4j.xml* or *log4j.properties*.

Logging

Otherwise, you can create your own log4j configuration file. This should be a customised version of the file *WEB-INF/classes/log4j.xml* inside *radrunner.war*, which is itself to be found inside *radrunner.ear*. Either:

1. Edit *WEB-INF/classes/log4j.xml* inside *radrunner.war*
2. Place your own version of *log4j.xml* somewhere on the network, then set the system property *log4j.configuration* to point to it (see [System Properties](#)).

The most likely desired change is to reset the general *Logging Level*, for example from DEBUG (the default, which logs every program step of consequence) to INFO (process information messages only) or UI (every internationalized message generated). Guidance on making these changes is provided inside the *log4j.xml* file provided with RADRunner. Such changes will provide less or more detailed logging, and may have a minor impact on system performance.

It is also possible to use log4j to customise the *Logging Categories*: the logging carried out for specific Java classes or packages. This can be used to provide very precise customisation of logging. See the log4j home page for details.

4.2. Log Message Format

The format of RADRunner log messages can be customised via the system properties:

- *com.rolemodellers.rim.logPattern*
- *com.rolemodellers.rim.logThreadPattern*

The default format is Comma-Separated Variable (CSV), which can be imported into most spreadsheets and databases:

- *com.rolemodellers.rim.logPattern=%-6r,%d{ISO8601},%15.15t,%-5p,%40.40c{3},%m%n*
- *com.rolemodellers.rim.logThreadPattern={0},{1},*

This corresponds to log messages of the form:

```
8802 ,2003-08-04 12:04:10,755, Thread-12,INFO , playwright.type.RMAction,RADRunner
Boss Actor,RADRunner 42 Role, Using transaction 1 in automated activity "Maintain Users"
```

An alternative, and more human-readable, format is:

- *com.rolemodellers.rim.logPattern=%-6r %d{ISO8601} [%15.15t] %-5p %40.40c{3} - %m%n*
- *com.rolemodellers.rim.logThreadPattern='{0}:{1}'*

This corresponds to log messages of the form:

```
429658 2003-08-04 12:25:57,334 [ Thread-12] INFO playwright.type.RMAction -
{RADRunner Boss Actor:RADRunner 42 Role} Using transaction 8 in automated activity
```

"Maintain Users"

5. Custom Logging

Logging in RADRunner is implemented via the open source framework log4j. However, it is possible to use other logging mechanisms - for instance, to integrate RADRunner logging with that carried out in other applications.

To implement this, all that is necessary is to create a new Java class which implements the following interface:

```
package com.rolemodellers.message;

public interface RMLogger {
    /**
     * Set up logging.
     */
    public boolean startLogging(String serverName);

    /**
     * Log a message.
     *
     * @param fromClass String - fully qualified name of the Java class that created this
     * @param thread String array - containing an actor name and a role name
     * @param levelStr String - the log priority
     * @param message String - an internationalized log message
     */
    public void logMessage(String fromClass, String[] thread, String levelStr, String message);

    /**
     * Release any resources used by the logger.
     */
    public boolean stopLogging();
}
```

The compiled form of your implementation should be placed on the classpath of the application server running RADRunner, and the system property *com.rolemodellers.rim.logger* set to the fully qualified class path of your new class (the default is *com.rolemodellers.message.RMLog4JLogger*).