"log4r"
A log4j-Comparable Logging Framework for ooRexx Applications

Rony G. Flatscher (Rony.Flatscher@wu-wien.ac.at), Wirtschaftsuniversität Wien

"The 2007 International Rexx Symposium", Tampa, Florida, U.S.A.

April 29th – May 3rd 2007.

Abstract: *While developing ooRexx applications programmers usually employ statements in the code that help debug the applications, gather (interesting) information about running those ooRexx programs and finally, report and log error conditions that may have risen while executing programs.*

The `log4r` framework has been devised after studying and experimenting with Apache's log4j logging framework. It attempts to apply some of the human-centric philosophy of Rexx, by simplyfying the framework as much as possible, but still allowing the creation of specialized appenders to extend the framework. The application of the log4r logging framework for (oo)Rexx programmers is quite simple, straight forward and therefore easy to use.

1 Introduction

A very important part of developing software is testing and debugging software. For systematic testing testunit frameworks have been devised for many languages, one for ooRexx was introduced in 2006 [Flat06]. While unit testing helps assure that routines and methods behave according to their specifications, and integration tests assert that the units are working together according the set-forth specification, this may not be enough.

While developing applications it may become important to learn and understand when which part of an application is being invoked and in case of anomalities to decide whether a program should be stopped or is able to continue to execute safely. Also monitoring deployed applications and learning about their execution paths and states needs logging information to be produced by the application.

In the context of the Apache organization an interesting logging framework has been devised and implemented for various languages, the original one which was created for the programming language Java is named 'log4j', "logging for Java" [W3LOG, W3L4J]. For 'log4j there exist numerous free introductions and tutorials [W3G02, W3M05, W3S01] including a manual of one of the original authors, C. Gülcü [W3G04].

This article introduces a "logging for Open Object Rexx (ooRexx), dubbed 'log4r'" which follows the `log4j` architecture (v. 1.2) quite closely. Of course ooRexx being an interpreted language possessing among other things a defined environment for executing programs and coupling them will differ in its implementation. In addition, compared to `log4j` a few enhancements and changes have been incorporated, like allowing layouts and filters to be named objects, which can be easily reused with different appenders.

This article introduces the logging architecture for Rexx coders. It first will document and explain how the `log4r` framework can be put to work for logging the execution of Rexx programs. This will be followed by an architectural overview and a description of the functionalities (and their configuration) that different classes make available.

As the logging framework can be configured with properties which can be explicitly defined in a property text-file, it is important to know the property keys and its values.

2 An Example of Putting 'log4r' to Work

Figure 2.2 depicts an ooRexx program which defines a class "Person", with a constructor (method "`init`"), a destructor (method "`uninit`"), the attribute methods "`familyName`", "`firstName`", "`salary`", a method "`string`" for creating a string with information about a person object, whenever ooRexx needs a string representation of a Person object and finally, a method "`increaseSalary`" to allow increasing the salary of persons. In the latter method the SYNTAX condition is intercepted, in case a wrong argument is supplied with which one cannot carry out arithmetics. The main program (at the top) will create an instance of the president of the RexxLA in 2007, Lee Peedin, with a due salary, that gets changed a few times, once with a non-numeric string.

The output is shown in figure 2.1. The program will run successfully and no one would note that in one particular invocation of the `increaseSalary` method, that method's code actually ran into a syntax condition!

For debugging and monitoring that program one could start to add `SAY` statements to log informative messages to the console, at least in the case that (e.g. syntax) errors occur. For debugging purposes one could display the received argument values. And if the flow of control is important for analysis and/or debugging, then trace statements might be in order, which print out the arrival in any method (e.g.

```
----------------- The beginning: -----------------
p=Peedin, Lee: 250000
-------------------- The end. --------------------
```

Figure 2.1: Output of Running the ooRexx Program in Figure 2.2.

```
say ' The beginning: '~center(50, "-") /* draw a line    */

p=.person~new("Peedin", "Lee", 250000) /* create a person*/
say "p="p~string           /* show the person's state    */
p~increaseSalary(12345.67) /* increase salary            */
p~increaseSalary("abc")    /* provoke an error           */
p~increaseSalary(-1000)    /* decrease salary            */

say ' The end. '~center(50, "-") /* draw a line          */

/* ============================================================ */
::class person          /* class "PERSON"                */

/* ---------------------------------------------------- */
::method init           /* method "INIT" (constructor)   */
  expose familyName firstName salary
  use arg familyName, firstName, salary

::method familyName     attribute
::method firstName      attribute
::method salary         attribute

/* ---------------------------------------------------- */
::method string         /* create a string rendering of a person  */
  expose familyName firstName salary
  return familyName"," firstName":" salary

/* ---------------------------------------------------- */
::method increaseSalary /* method to increase the salary */
  expose salary
  parse arg raise

  signal on syntax      /* in case arithmetic creates a condition */
  salary=salary+raise
  return
syntax:                 /* just there to let the program continue */

/* ---------------------------------------------------- */
::method uninit         /* optional destructor method    */
```

Figure 2.2: '`t0.rex`" - A Simple ooRexx Program Causing an Unnoticed Syntax Error.

giving the method's name and the object for which it got invoked) or routine.

But what, if the program does not run from a console? In such cases those log messages could not be read and studied at all!

Another question one could ask is how to treat that log code which produces the error, warn, debug, trace messages, if an application is supposed to be deployed. Depending on the size of an application it may be very time consuming (and error prone) to physically delete the log statements. If it was possible to leave the log messages in the code, but turn off the outputting of these messages, then such an application could be rolled out, *without* removing any log message in the code!

Having log messages in the deployed application and a means available to activate the messages and log them to the console, or to files, then monitoring and analyzing deployed applications becomes possible! The `log4r` framework allows for that for ooRexx programs. Figure 2.3 enhances the program in figure 2.2 with numerous log

```
say ' The beginning: '~center(50, "-") /* draw a line    */
call load_log4r          /* load the 'log4r' framework    */
l=.logManager~getLogger("rgf.test") /* get/create a logger named 'rgf.test'   */

parse arg logLevel      /* retrieve logLevel, if supplied */
if logLevel="" then l~logLevel="OFF"      /* do not show any log messages     */
               else l~logLevel=logLevel   /* set logLevel to argument's value */
l~debug("just created a logger named 'rgf.test':" pp(l~string))

parse source s          /* get source information        */
l~trace("source:" pp(s))

p=.person~new("Peedin", "Lee", 250000) /* create a person*/
say "p="pp(p~string)     /* pp() defined in 'log4r' framework   */
p~increaseSalary(12345.67) /* increase salary            */
p~increaseSalary("abc")    /* provoke an error           */
p~increaseSalary(-1000)    /* decrease salary            */
say ' The end. '~center(50, "-") /* draw a line          */
l~trace("end of program.")

/* ================================================================= */
::class person          /* class "PERSON"                */

/* ----------------------------------------------------- */
::method init           /* method "INIT" (constructor)   */
  expose familyName firstName salary

  l=.logManager~getLogger("rgf.test")     /* get logger  */
  l~trace("method 'init'")

  use arg familyName, firstName, salary
  l~debug("method 'init' - created the following person:" pp(self~string))

  if salary>10000 then  -- warn about something
     l~warn("method 'init' - salary quite high:" salary)

::method familyName     attribute
::method firstName      attribute
::method salary         attribute

/* ----------------------------------------------------- */
::method string
  expose familyName firstName salary

  .logManager~getLogger("rgf.test")~trace("method 'string'")
  return familyName"," firstName":" salary

/* ----------------------------------------------------- */
::method increaseSalary /* method to increase the salary */
  expose salary

  l=.logManager~getLogger("rgf.test")
  l~trace("method 'increaseSalary'")

  parse arg raise
  l~debug("method 'increaseSalary', received="pp(raise))
  signal on syntax      /* in case arithmetic creates a condition    */
  salary=salary+raise
  l~debug("method 'increaseSalary', new salary="pp(salary))
  return
syntax:
  l~error("method 'increaseSalary', exception has occurred!", condition("O"))

/* ----------------------------------------------------- */
::method uninit
  .logManager~getLogger("rgf.test")~trace("method 'uninit'")
  .logManager~getLogger("rgf.test")~debug("method 'uninit' running for person:" pp(self))
```

Figure 2.3: 't1.rex" - A Simple ooRexx Program Containing 'log4r' Log Statements.

statements for the purpose of logging errors, warnings (if a salary seems to be too high), debug and trace messages. All the applied changes are highlighted with a light violet background in figure 2.3. Running it yields the same result and output as above (cf. figure 2.1 above).

However, now it has become possible with the help of log4r to get all the errors that

```
----------------- The beginning: -----------------
p=[Peedin, Lee: 250000]
    1: 0.301000 [rgf.test] ERROR - method 'increaseSalary', exception has occurred!
                        ADDITIONAL..[an Array] containing 1 item(s)
                                         --> [abc]
                        CODE........[41.1]
                        CONDITION...[SYNTAX]
                        DESCRIPTION.[]
                        ERRORTEXT...[Bad arithmetic conversion]
                        INSTRUCTION.[SIGNAL]
                        MESSAGE.....[Nonnumeric value ("abc") used in arithmetic operation]
                        POSITION....[63]
                        PROGRAM.....[F:\test\t1.rex]
                        PROPAGATED..[0]
                        RC..........[41]
                        TRACEBACK...[a List] containing 1 item(s)
                                         --> [    63 *-* salary=salary+raise]

-------------------- The end. --------------------
```

Figure 2.4: Output of Running the ooRexx Program in Figure 2.3 with "rexx t1.rex error".

got logged by invoking it as "rexx t1.rex error", yielding an output like in figure 2.4, documenting exactly the error by displaying all the information supplied by ooRexx.

The log4r framework defines six log levels which are internally represented as numbers: TRACE < DEBUG < INFO < WARN < ERROR < FATAL. Running at a log level ("cateory") of ERROR will only process log messages that have a log level of ERROR or higher.

In the program of figure 2.3 a class named LogManager is used to get a logger, by supplying the logger's name as an argument to the getLogger message. The returned logger accepts messages by the same name as the desired log level: trace, debug, info, warn, error, fatal. These methods accept one string argument, and an optional second argument which would be a condition object containing all condition information. Whether these log messages get processed or not depends on the logger's setting of logLevel: only messages at the given log level or with a higher log level will get processed.

The processed log messages get forwarded to each appender in the logger's appender queue. Therefore a processed log message may be further processed by multiple appenders. Each appender can be independently set to a log level threshold

```
say ' The beginning: '~center(50, "-") /* draw a line    */

call load_log4r         /* load the 'log4r' framework    */

l=.logManager~getLogger("rgf.test") /* get (create?) a logger named 'rgf.test'   */

   /* configure the logger a little bit:  */
app=.FileAppender~new("test.rgf.app")        /* create an appender         */
app~layout=.HTMLLayout~new("test.rgf.layout")/* create a layout and assign it to appender */
app~fileName="test_"date("S")".html"         /* set filename               */
app~append=.false                            /* replace an existing file   */
l~addAppender(app)                           /* add appender to logger     */

.local~test.rgf.logger=l   /* save logger in .local, even easier to refer to  */

l~debug("appender created at runtime:" pp(app~name":" getLogLevelAsString(app~threshold)))

parse arg logLevel
if logLevel="" then l~logLevel="OFF"      /* do not show any log messages     */
               else l~logLevel=logLevel   /* set logLevel to argument's value */

... cut ...
```

Figure 2.5: '`t2.rex`*" – Adapting Program in Fig. 2.3 to Create a HTML File.*

value, which enables such an appender to ignore all supplied log messages with a lower log level.

Changing the program of figure 2.3 a little bit will allow to add an additional appender to the logger retrieved with the help of the `LogManager` class. The new appender will save log messages in HTML form to the specified file.

The necessary changes are given in figure 2.5, in which the code additions are highlighted. Running that program from the command line with "`rexx t2.rex trace`" will process all log messages, which will be shown on the console (cf. figure 2.6) and written into a file formatted as HTML. The HTML file can be viewed with any browser, figure 2.7 on p. 8 depicts how the MS Internet Explorer will render the HTML text.

Besides adding the log messages and retrieving the logger with the help of the class, not much had to be done by the ooRexx code to achieve this log processing.

Studying the output briefly (e.g. figure 2.7), one can see that the log messages are sequentially numbered, the date and time of the log message is given, the elapsed time in the application since the logging has started, the name of the logger used to process the log message and finally the logged message itself. In the case of a condition the content of the condition object is displayed, the output sorted by the key values and wherever array or lists are part of the information stored in the condition object, that content is shown as well.

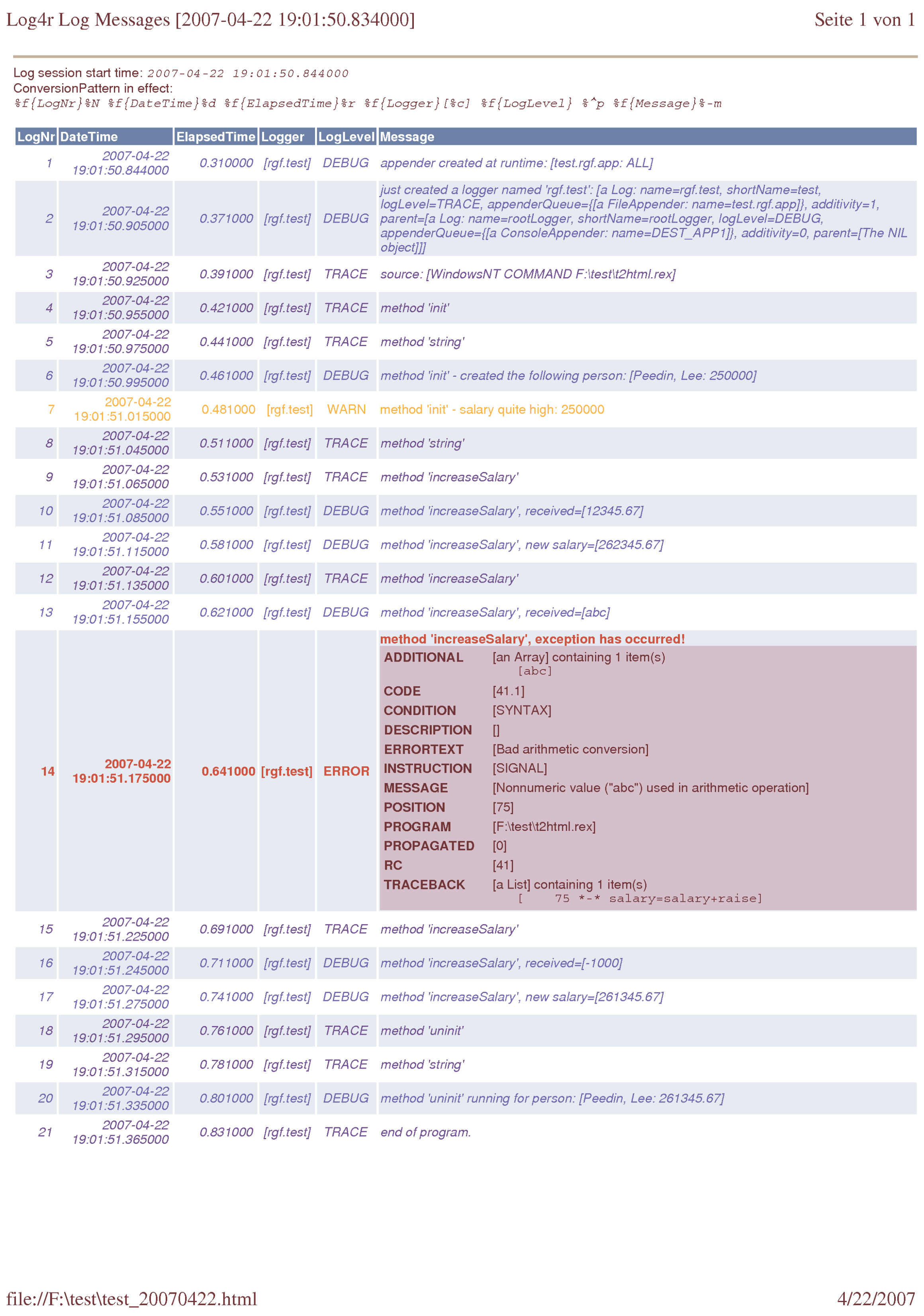
In principle a programmer is able to control all the loggers that are maintained via

```
----------------- The beginning: -----------------
    1: 0.310000 [rgf.test] DEBUG - appender created at runtime: [test.rgf.app: ALL]
    2: 0.371000 [rgf.test] DEBUG - just created a logger named 'rgf.test': [a Log: name=rgf.test,
shortName=test, logLevel=TRACE, appenderQueue={[a FileAppender: name=test.rgf.app]}, additivity=1,
parent=[a Log: name=rootLogger, shortName=rootLogger, logLevel=DEBUG, appenderQueue={[a ConsoleAppender:
name=DEST_APP1]}, additivity=0, parent=[The NIL object]]]
    3: 0.391000 [rgf.test] TRACE - source: [WindowsNT COMMAND F:\test\t2html.rex]
    4: 0.421000 [rgf.test] TRACE - method 'init'
    5: 0.441000 [rgf.test] TRACE - method 'string'
    6: 0.461000 [rgf.test] DEBUG - method 'init' - created the following person: [Peedin, Lee: 250000]
    7: 0.481000 [rgf.test] WARN  - method 'init' - salary quite high: 250000
    8: 0.511000 [rgf.test] TRACE - method 'string'
p=[Peedin, Lee: 250000]
    9: 0.531000 [rgf.test] TRACE - method 'increaseSalary'
   10: 0.551000 [rgf.test] DEBUG - method 'increaseSalary', received=[12345.67]
   11: 0.581000 [rgf.test] DEBUG - method 'increaseSalary', new salary=[262345.67]
   12: 0.601000 [rgf.test] TRACE - method 'increaseSalary'
   13: 0.621000 [rgf.test] DEBUG - method 'increaseSalary', received=[abc]
   14: 0.641000 [rgf.test] ERROR - method 'increaseSalary', exception has occurred!
                      ADDITIONAL..[an Array] containing 1 item(s)
                                       --> [abc]
                      CODE........[41.1]
                      CONDITION...[SYNTAX]
                      DESCRIPTION.[]
                      ERRORTEXT...[Bad arithmetic conversion]
                      INSTRUCTION.[SIGNAL]
                      MESSAGE.....[Nonnumeric value ("abc") used in arithmetic operation]
                      POSITION....[75]
                      PROGRAM.....[F:\test\t2html.rex]
                      PROPAGATED..[0]
                      RC..........[41]
                      TRACEBACK...[a List] containing 1 item(s)
                                       --> [    75 *-* salary=salary+raise]

   15: 0.691000 [rgf.test] TRACE - method 'increaseSalary'
   16: 0.711000 [rgf.test] DEBUG - method 'increaseSalary', received=[-1000]
   17: 0.741000 [rgf.test] DEBUG - method 'increaseSalary', new salary=[261345.67]
   18: 0.761000 [rgf.test] TRACE - method 'uninit'
   19: 0.781000 [rgf.test] TRACE - method 'string'
   20: 0.801000 [rgf.test] DEBUG - method 'uninit' running for person: [Peedin, Lee: 261345.67]
-------------------- The end. --------------------
   21: 0.831000 [rgf.test] TRACE - end of program.
```

Figure 2.6: Output of Running the ooRexx Program in Figure 2.5 with "`rexx t2.rex trace`".

the `log4r` framework. This allows for instance the controlled activation of log message processing for applications that third parties may have created and that employ the `log4r` framework themselves. As an example any programmer can take advantage of the `rgf.Socket` class which is part of `log4r` (to enable the creation of the TelnetAppender, which allows using telnet to look up the processed log messages on a remote computer). Its logger is named "`rgf.sockets`". If someone wants to trace the processing of messages sent to `rgf.socket` instances, then one would need to retrieve that logger and set its log level to `TRACE`. Whenever tracing should be shut off, the logger's log level can be reset.



Log4r Log Messages [2007-04-22 19:01:50.834000]   Seite 1 von 1

Log session start time: `2007-04-22 19:01:50.844000`
ConversionPattern in effect:
`%f{LogNr}%N %f{DateTime}%d %f{ElapsedTime}%r %f{Logger}[%c] %f{LogLevel} %^p %f{Message}%-m`

LogNr	DateTime	ElapsedTime	Logger	LogLevel	Message
1	2007-04-22 19:01:50.844000	0.310000	[rgf.test]	DEBUG	appender created at runtime: [test.rgf.app: ALL]
2	2007-04-22 19:01:50.905000	0.371000	[rgf.test]	DEBUG	just created a logger named 'rgf.test': [a Log: name=rgf.test, shortName=test, logLevel=TRACE, appenderQueue={[a FileAppender: name=test.rgf.app]}, additivity=1, parent=[a Log: name=rootLogger, shortName=rootLogger, logLevel=DEBUG, appenderQueue={[a ConsoleAppender: name=DEST_APP1]}, additivity=0, parent=[The NIL object]]]
3	2007-04-22 19:01:50.925000	0.391000	[rgf.test]	TRACE	source: [WindowsNT COMMAND F:\test\t2html.rex]
4	2007-04-22 19:01:50.955000	0.421000	[rgf.test]	TRACE	method 'init'
5	2007-04-22 19:01:50.975000	0.441000	[rgf.test]	TRACE	method 'string'
6	2007-04-22 19:01:50.995000	0.461000	[rgf.test]	DEBUG	method 'init' - created the following person: [Peedin, Lee: 250000]
7	2007-04-22 19:01:51.015000	0.481000	[rgf.test]	WARN	method 'init' - salary quite high: 250000
8	2007-04-22 19:01:51.045000	0.511000	[rgf.test]	TRACE	method 'string'
9	2007-04-22 19:01:51.065000	0.531000	[rgf.test]	TRACE	method 'increaseSalary'
10	2007-04-22 19:01:51.085000	0.551000	[rgf.test]	DEBUG	method 'increaseSalary', received=[12345.67]
11	2007-04-22 19:01:51.115000	0.581000	[rgf.test]	DEBUG	method 'increaseSalary', new salary=[262345.67]
12	2007-04-22 19:01:51.135000	0.601000	[rgf.test]	TRACE	method 'increaseSalary'
13	2007-04-22 19:01:51.155000	0.621000	[rgf.test]	DEBUG	method 'increaseSalary', received=[abc]

14	**2007-04-22 19:01:51.175000**	**0.641000**	**[rgf.test]**	**ERROR**	**method 'increaseSalary', exception has occurred!**
					ADDITIONAL [an Array] containing 1 item(s)
					`[abc]`
					CODE [41.1]
					CONDITION [SYNTAX]
					DESCRIPTION []
					ERRORTEXT [Bad arithmetic conversion]
					INSTRUCTION [SIGNAL]
					MESSAGE [Nonnumeric value ("abc") used in arithmetic operation]
					POSITION [75]
					PROGRAM [F:\test\t2html.rex]
					PROPAGATED [0]
					RC [41]
					TRACEBACK [a List] containing 1 item(s)
					`[    75 *-* salary=salary+raise]`

15	2007-04-22 19:01:51.225000	0.691000	[rgf.test]	TRACE	method 'increaseSalary'
16	2007-04-22 19:01:51.245000	0.711000	[rgf.test]	DEBUG	method 'increaseSalary', received=[-1000]
17	2007-04-22 19:01:51.275000	0.741000	[rgf.test]	DEBUG	method 'increaseSalary', new salary=[261345.67]
18	2007-04-22 19:01:51.295000	0.761000	[rgf.test]	TRACE	method 'uninit'
19	2007-04-22 19:01:51.315000	0.781000	[rgf.test]	TRACE	method 'string'
20	2007-04-22 19:01:51.335000	0.801000	[rgf.test]	DEBUG	method 'uninit' running for person: [Peedin, Lee: 261345.67]
21	2007-04-22 19:01:51.365000	0.831000	[rgf.test]	TRACE	end of program.

file://F:\test\test_20070422.html   4/22/2007

Figure 2.7: Output of Running the ooRexx Program in Figure 2.5 with '`rexx t2.rex trace`".

3 The 'log4r' Architecture

The `log4r` framework creates an environment in which a class named `LogManager` is available that allows applications to retrieve (get) loggers by name using its class method `getLogger`, which expects a logger name as an argument. In the case that the logger has not been created yet, `.LogManager` creates one, stores it for later retrieval and returns it to the requesting application.

A logger is an instance of one of the following classes: `Log`, `SimpleLog`, or `NoOp`.[1] All of these classes understand the log messages `trace`, `debug`, `info`, `warn`, `error`, and `fatal`. In addition the following methods are implemented returning either `.true` or `.false`, depending whether processing of log messages at that particular log level is active:[2] `isTraceEnabled`, `isDebugEnabled`, `isInfoEnabled`, `isWarnEnabled`, `isErrorEnabled` and `isFatalEnabled`. For logging messages no other methods are needed.

Each logger of type `Log` has a name and may have one or more appenders in its appender queue, to which log messages are forwarded for processing. Each appender can process a log message according to an optional layout which is used to format the log message. Each appender can be configured for a threshold log level and optionally apply filters which ultimately decide, whether a received log message will be processed by it. Figure 3.1 depicts these relationships.

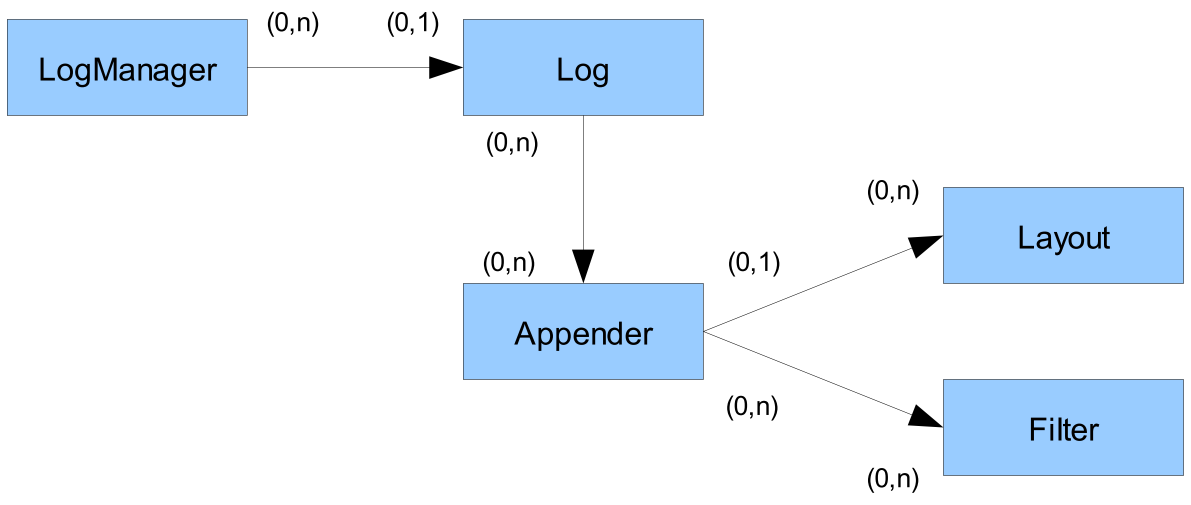


Figure 3.1: Relationships (with Participation Constraints) Among the Classes `LogManager`, `Log`, `Appender`, `Layout`, and `Filter`.

[1] The class that gets used is determined by the environment symbol `.log4r.config.LoggerFactoryClassName`. It contains the name of the logger class (`"Log"`, `"SimpleLog"`, `"NoOpLog"`) to be used.

[2] These methods are meant for situations where the creation of log messages may be very time or resource consuming, allowing to determine whether a log message of a particular log level would be processed by the logger, *before* creating the message: the result `.true` indicates that a message of that log level would be processed, `.false` that it would not.

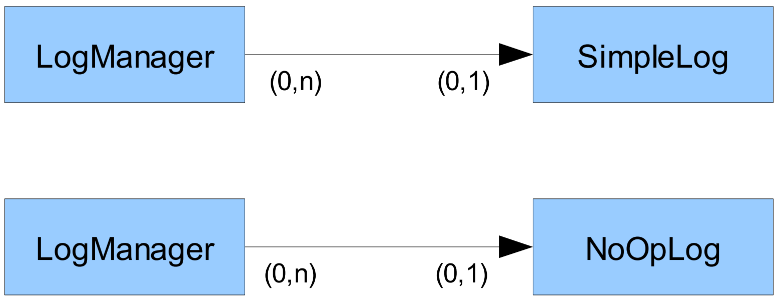


Figure 3.2: Relationships (with Participation Constraints) Between `LogManager` and the `SimpleLog`, respectively the `NoOpLog` Classes.

Figure 3.2 depicts the relationships available for loggers of type `SimpleLog` and `NoOpLog`, which both do not relate to appenders.

The following subsections will introduce and briefly characterize the classes that are building the log4r infrastructure, the logger classes, the appender classes, the filter classes and the layout classes.[3]

3.1 The Infrastructural 'log4r' Classes

Figure 3.3 below shows the infrastructural classes `LogLog`, `LogManager`, `Log4r.Properties`, and `Log4r.Timing` with the methods they define. Class methods are highlighted in a bold font, the built-in ooRexx class `Directory` and its methods are shown in a grey color.

3.1.1 The “LogLog” Class

The `LogLog` class allows all the `log4r` framework classes to produce log messages themselves, supplying the message (a string), and optionally a second argument which can be a condition object.

The programmers can use the `debug` class method (outputs to the standard output stream using the ooRexx built-in `.output` monitor object), the warn and error class methods (output to the stadard error stream using the built-in `.error` monitor object).

Debug messages are by default inhibited. This is controlled by the logical values `.true` or `.false` stored in the ooRexx `.local` environment with the name: "`log4r.config.LogLog.Debug`". Setting this environment entry to `.true` will cause debug messages to be shown, otherwise they will be supressed. Warning and error

[3] The ooRexx classes `rgf.Socket` and `rgf.ServerSocket` that were created for realizing the TelnetAppender class are shown in the appendix.

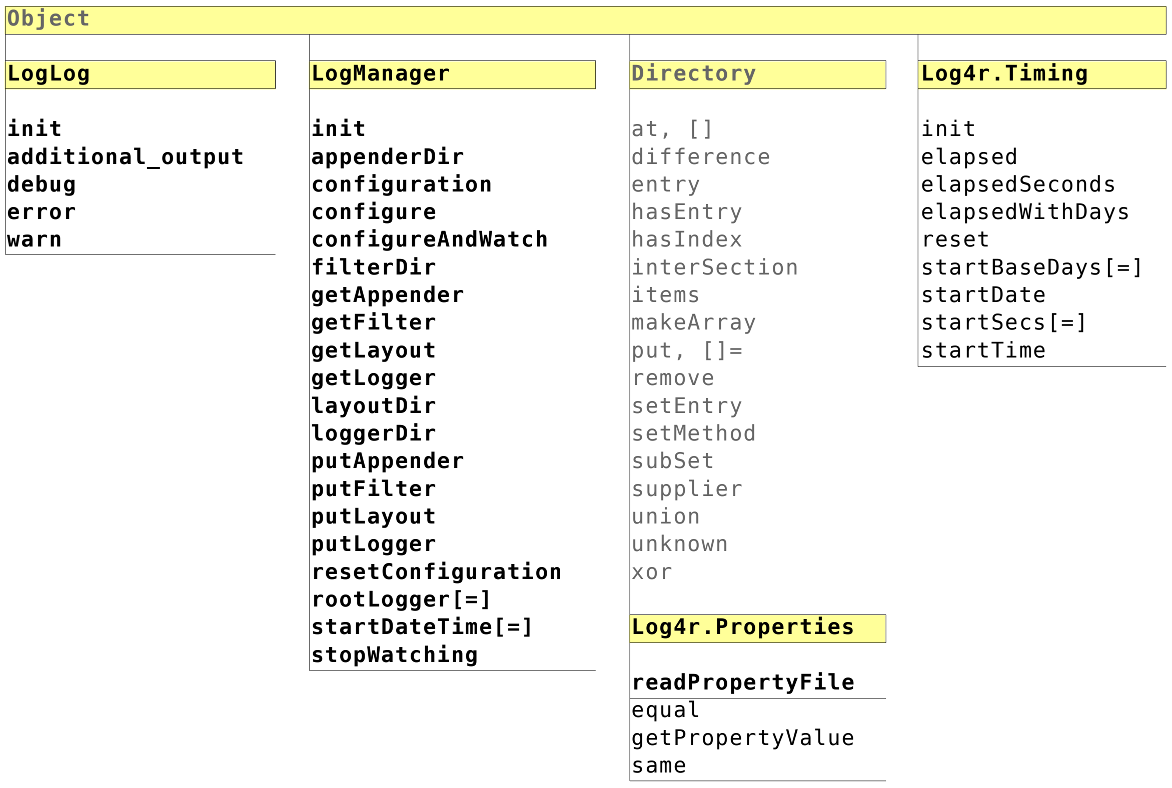


Figure 3.3: The Infrastructural 'log4r' Classes (Class Methods Are in Bold).

messages will be shown, independently of the setting of the aforementioned environment entry.

However, if no log messages should be shown whatsoever, then one can do so, by setting an environment entry named "`log4r.config.LogLog.QuietMode`" to `.true`.

3.1.2 The "LogManager" Class

The `LogManager` class possesses only class methods. It is not meant to be used to create instances, but rather serves as a utility class. It can be accessed by any ooRexx program using the environment symbol `.LogManager`.

Its most important class method is `getLogger`, which expects a name for a logger. It will use that name to look up the logger in its `loggerDir` directory and returns the logger object, if available. Otherwise a logger object is created, using the class denoted by name in the environment symbol `.log4r.config.LoggerFactoryClassName`. saved in its `loggerDir` directory and thereafter returned.

The LogManager can also serve appenders (`getAppender`), layouts (`getLayout`), and filters (`getFilter`) by name. If the objects are not available, and an optional class object is supplied as the second argument, then a default instance is created, stored with the supplied name in the appropriate directory object and returned.

Another important service the `LogManager` class carries out, is the processing of (optional) `log4r` property files, which may contain definitions for runtime options, default values for `SimpleLogger`, values for configuring named loggers, named appenders, named layouts, and named filters. `.LogManager` will be able to monitor such a configuration file for changes and apply those changes to a running system, if the configuration is set up for it. This montoring for file changes is carried out if the environment symbol "`.log4r.config.configFileWatchInterval`" stores a number which is larger than 0 seconds.[4]

3.1.3 The “log4r.Properties” Class

The `Log4r.Properties` class is a specialisation of the built-in class `.Directory`. It allows for reading `log4r` property files. The property keys will be stripped of leading and trailing blanks and put into uppercase. The `getPropertyValue` method allows retrieving property entries, optionally supplying a default value which should be returned in the case that there is no entry for the supplied key. In addition it allows to compare two properties objects to determine whether they are `equal` (ignoring leading and trailing blanks in the values) or identical (method `same`), i.e., comparing values byte by byte.

This class is responsible for parsing property files and collecting their definitions.

3.1.4 The “log4r.Timing” Class

The `Log4r.Timing` class is responsible for keeping an elapsed time counter. As ooRexx does not have any statistics available for this kind of information, a surrogate had to be created.

Whenever the `log4r` framework is initialized one (central) instance of this class is created and used for measuring the elapsed time from that moment on. This information is then added to the log message directory object such that it can be used to indicate the elapsed time for that particular message since the start of the `log4r` framework.

3.2 The Logger 'log4r' Classes

Usually, the only class a user of the `log4r` framework is confronted with, is of the type "`Log`". Its instances are called "loggers" and are usually maintained by the `LogManager` class, which one uses to retrieve a specific logger at runtime by supplying the logger's name with the `getLogger` message.

[4] If a file has not been watched for changes, after changing the watch interval to a number greater than 0, the programmer needs to send the `.LogManager` the `watchAndConfigure` message to (re-) start watching.

To create log messages the programmer would send a logger one of the following messages, which are named after the log level they represent: `trace`, `debug`, `info`, `warn`, `error` (error, but continuing with processing is possible), `fatal` (fatal error, program cannot continue to execute). Each of these methods expects a message as an argument (a string) and may have one or two additional arguments. The second argument, if given, is either the condition object, or may be a string or a directory object and is then dubbed the "additional" argument. If the second argument is a condition object, then the "additional" argument would be supplied as the third argument.

The attribute `logLevel` determines, which log messages the logger will process: it will only process log messages that are of the same log level or higher. The relation between the different levels of log messages is as follows: `ALL` < `TRACE` < `DEBUG` < `INFO` < `WARN` < `ERROR` < `FATAL` < `OFF`.[5] To assign a new value, assign `logLevel` a string supplying the name of the log level.

Sometimes it may be the case that creating the log message is time and/or resource consuming, such that the programmer wishes to carry it out only, if such a produced log messages is processed by the logger. To help determine this, the boolean methods `isTraceEnabled`, `isDebugEnabled`, `isInfoEnabled`, `isWarnEnabled`, `isErrorEnabled`, `isFatalEnabled` have been created. These methods will return `.true`, if the processing of that log level is active for the logger, `.false` else.

The `Log` class is the standard and most powerful of the available logger classes. It gets used, if the `log4r` framework at start up finds the standard property file "`log4r.properties`", or if a custom property file was processed that defines the string "`Log`" with the key "`log4r.config.LoggerFactoryClassName`".

The `SimpleLog` class does not employ any appenders and is capable of simple log message processing, sending the message to the console. It is used, if only the standard property file "`simplelog4r.properties`" is found, or if a custom property file was processed that defines the string "`SimpleLog`" with the key "`log4r.config.LoggerFactoryClassName`".

The `NoOpLog` class is the "null operation Log class" and does not process any of the received log messages. This allows deployed applications to run and send off log messages that do not get processed, instead the methods will return immediately. All methods testing for processing log messages of a certain level will return `.false` to inhibit the creation of time/resource hungry log messages. It is this class that allows keeping log messages in the applications. This logger class is used, if none of

[5] Each log level is represented internally by a number allowing to carry out the comparisons. The log levels `ALL` and `OFF` are special, in that `ALL` is set to `0`, the lowest possible log level number, and `OFF` is set to a number which is higher than `FATAL`, inhibiting the processing of log messages at the fatal level.

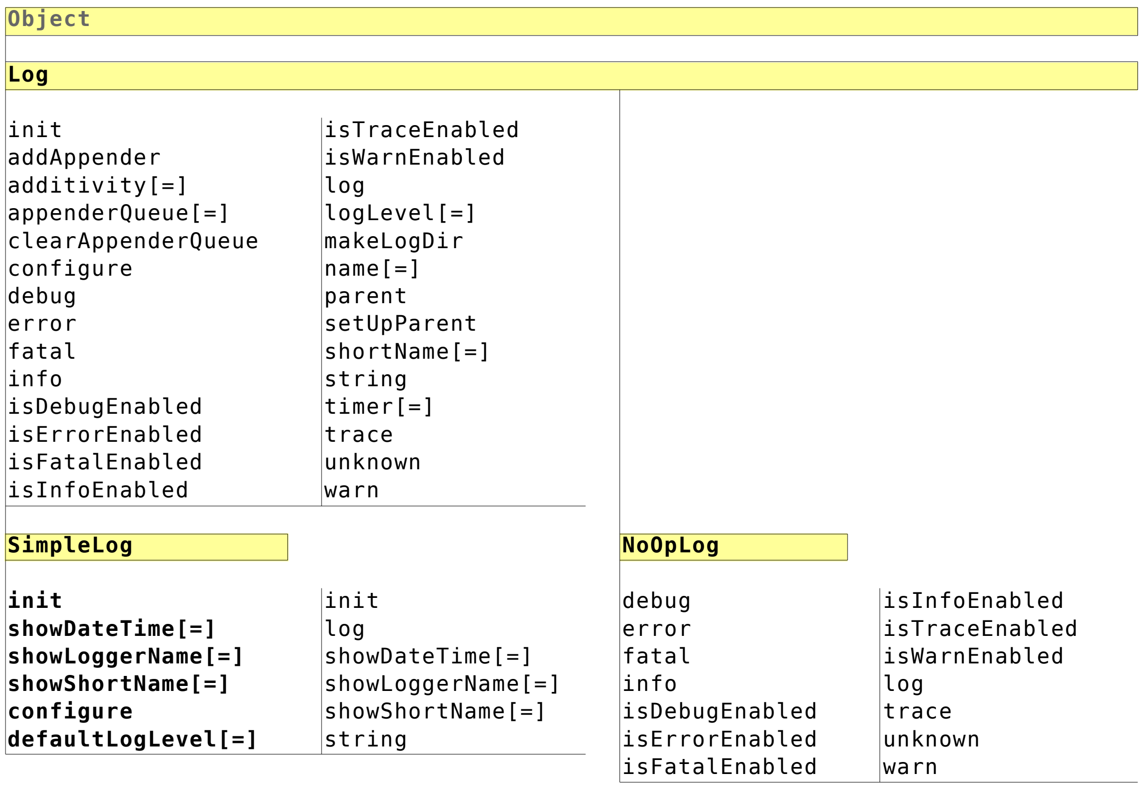


Figure 3.4: The Logger Classes `Log`, `SimpleLog`, and `NoOpLog` (Class Methods Are in Bold)

the standard property files "`log4r.properties`" and "`simplelog4r.properties`" could be found, or if a custom property file was processed that defines the string "`NoOpLog`" with the key "`log4r.config.LoggerFactoryClassName`".

It is possible to globally define a minimum (threshold) log level for processing log messages, independent of the settings of existing loggers. This is done by defining an environment symbol named "`.log4r.config.disable`" and setting it to the minimum (threshold) log level: all log messages at a lower level will not be processed by any loggers, even if loggers existed that would have an appropriate log level. Setting this value to `OFF` in effect stops processing log messages for the entire `log4r` framework! However, this runtime setting is only respected, if a second environment symbol named "`.log4r.config.disableOverride`" exists and is set to `.false`. This way, it becomes possible to temporarily allow to remove the effect of the "`.log4r.config.disable`" setting, by changing the value of the environment symbol "`.log4r.config.disableOverride`" to `.true`.[6]

The processing of log messages is by default synchroneous, i.e., an application

[6] These changes can be done at runtime, either by programmatically changing the value of the environment symbols (entries in `.local`) or indirectly by changing the properties file entry relating to the global configuration of the `log4r` framework while that file is being watched.

sending a log message will be halted, until the log message has been processed in full. Sometimes, in very time critical applications it may be desirable to have the log messages processed asynchroneously, i.e. control returns immediately upon dispatching the message to the application. This feature is controlled by the environment symbol "`.log4r.config.asyncMode`", which by default is set to `.false`. Changing it to `.true` will switch to asynchroneous processing of log messages, making sure that the log messages are still processed in the sequence they were received.

Figure 3.4 above depicts the class hierarchy and the methods the logger classes inherit and implement.

3.2.1 The “Log” Class

The `Log` class is the most versatile of the logger classes and serves as the superclass for the `SimpleLog` and `NoOpLog` classes.

If the name of a logger has dots in it, then these dots delimit the "components" of the logger's name. It is possible that there are different loggers with different names, but have a common "trunk", i.e. they share one or more of the component names, if one reads the logger's name from left to right. Loggers that share common components, but have fewer components than the others, are called "parents" of these other loggers. It is possible to have the parent loggers process the same log messages their "descendants" have received. There exists a single "rootLogger" in the framework which serves as the root parent for any logger in the system.[7]

Loggers of this class process all received log messages with a log level being equal or higher than their `logLevel` attribute. They create a directory object for each received log message, add date, time and timing information to it and then sends that log message directory to each of the appenders stored in their `appenderQueue` attribute. In the case that no appenders are defined, then the class will simply send the log message to the console.

In addition, if the attribute `additivity` is set to `.true`, then the message directory object is sent to the logger's "parent", if it exists, otherwise it gets sent to the so called `rootLogger` for further processing.

3.2.2 The “SimpleLog” Class

The `SimpleLog` class, as its named suggests, simply outputs the received log messages by optionally displaying the date and time, the name of the logger

[7] The `rootLogger` is created by the log4r framework at start up and by default employes a `ConsoleAppender` object which sends the log messages to the console. It can be individually configured like any other logger.

instance in its full form or just the last component of the dotted name.

No appenders are employed for processing log messages.

3.2.3 The “NoOpLog” Class

The `NoOpLog` class represents the "no operation logger". It will allow to receive log messages, but does not process them and will return immediately. Any application using the `log4r` framework can use this class in deployed versions of their applications, such that no log messages are processed at a client site.

3.3 The Appender 'log4r' Classes

Loggers of type `Log` forward a log message directory object to "appenders" to carry out the "appending" of messages to whatever destination the appender got set up for and is capable of utilizing. An appender may serve more than one logger.

Appenders allow defining specific layout objects for formatting the messages and may employ filters which can be used to process just specific log messages. Each appender can be configured with a threshold (minimum) log level, independently of the logger it serves. This way appenders could ignore log messages sent to it for processing.

A logger may use more than one appender, e.g. one appender which appends the log message to the console (class `ConsoleAppender`), another one which appends the same log message to a file (class `FileAppender`), yet another one which appends the very same log message to a telnet port (class `TelnetAppender`). This way it is possible to create many different collections of the log messages from one run of an application. The initial example in this article has excercised this feature by creating a `FileAppender` which gets the log messages formatted as HTML in addition to the messages sent to the console (via the `rootLogger`), cf. figure 2.5 on p. 6, causing in addition to the console output (cf. figure 2.6 on p. 7) the creation of a HTML file containing the same log messages (cf. figure 2.7 on p. 8).

Figure 3.5 depicts the class hierarchy and the methods the appender classes inherit and implement.

The following subsections will sometimes explicitly explain some appender attributes, such that the reader becomes able to understand and set these very same values via configuration files or configure appenders at run time for that matter.

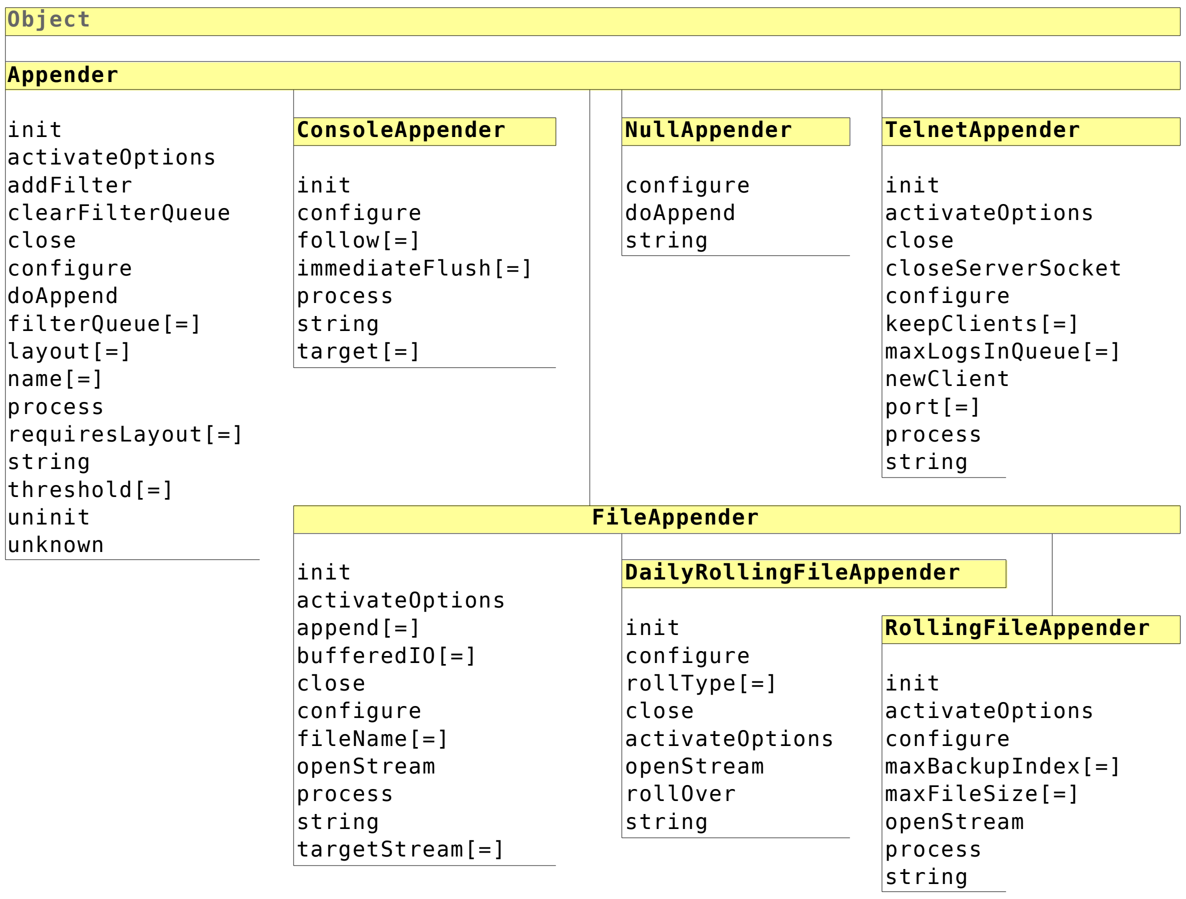


Figure 3.5: The Appender Class Hierarchy.

3.3.1 The “Appender” Class

The `Appender` class defines the attributes and methods that all specialized appender classes will possess.

The configurable attribute `threshold` allows to determine the minimum log level that an appender accepts for processing. To assign a new value, assign a string supplying the name of the log level: `"ALL"`, `"TRACE"`, `"DEBUG"`, `"INFO"`, `"WARN"`, `"ERROR"`, `"FATAL"`, or `"OFF"`. By default an appender accepts `"ALL"` log level messages it receives from a logger.

3.3.2 The “ConsoleAppender” Class

The `ConsoleAppender` class specializes the `Appender` class and appends log messages to the console.

The attribute `immediateFlush` is set to `.true` by default and causes the log messages to be immediatly written to the console, such that no buffering takes place.

The attribute `target` accepts either the standard output (string values `"stdout"`, `"output"`) or standard error (string values `"stderr"`, `"error"`) file as an argument.

The attribute `follow` (values `.true` or `.false`) determines, whether an attempt to change the `target` attribute value should be followed (accepted) or not.

3.3.3 The “FileAppender” Class

The `FileAppender` class specializes the `Appender` class and defines the attributes and methods that its specialized appender classes will possess. It appends the received log messages to a file.

The attribute `append` (values `.true` or `.false`) specifies, whether log messages should be appended to an existing file or whether an existing file would be truncated before appending log messages to it.

The attribute `bufferedIO` (values `.true` or `.false`) specifies, whether log messages should be appended immediately to an existing file, or whether buffering would be allowed.

The attribute `fileName` (any valid file name, may contain drive and path, if the filesystem supports it) specifies the name of the file to which the log messages should be written to.

3.3.4 The “RollingFileAppender” Class

The `RollingFileAppender` class specializes the `FileAppender` class.

It allows to automatically create new log files, in the case the current one has exceeded the value of the configurable attribute `maxFileSize`. The `maxFileSize` value (default value: `10MB`) can be given as a pure number of bytes or a number followed by one of: `kb` (kilo byte), `mb` (mega byte), `gb` (giga byte), or `tb` (tera byte).

The attribute `maxBackupIndex` (default value: `1`) determines how many generations of log files are kept. If this number is exceeded, then the oldest log file will be deleted, the existing ones renamed and a new file will be used to send the log messages to. If `maxBackupIndex` is set to the value `0`, then no backup log files are created.

3.3.5 The “DailyRollingFileAppender” Class

The `DailyRollingFileAppender` class specializes the `FileAppender` class.

The configurable attribute `rollType` allows to determine the roll over type, which causes the creation of a new log file to be used to append the log messages to. The filename gets the creation date (formatted as: `".YYY-MM-DD-"`) and time (formatted as:

`"HH_MM_SS_"`) and the name of the `rollType` appended to it.

`rollType` may denote one of the following roll over types:

- `"MINUTE"`: a new log file will be created and used at every full minute,
- `"HOUR"`: a new log file will be created and used at every full-hour,
- `"HALF_DAY"`: a new log file will be created and used at noon ("`12:00`") and at midnight ("`00:00`"),
- `"DAY"`: a new log file will be created at midnight of each day ("`00:00`"),
- `"WEEK"`: a new log file will be created each Monday at midnight ("`00:00`"),
- `"MONTH"`: a new log file will be created at the 1st of each month at midnight ("`00:00`").

3.3.6 The “NullAppender” Class

The `NullAppender` class specializes the `Appender` class and defines a "no operation appender". It can be used in cases, where (temporarily) an appender should not process log messages.

3.3.7 The “TelnetAppender” Class[8]

The `TelneAppender` class specializes the `Appender` class and allows dispatching log messages to a TCP/IP socket, which can be addressed via the telnet application from the same machine or another one. This appender enables the monitoring of log messages on remote computers. The implementation just reacts upon the receipt of the `CTL-C` or `CTL-D` characters, closing the respective client socket. Therefore deploying this appender is safe as it is not possible to use it as a backdoor or to attack a computer via the port that gets served.

The attribute `port` is set to the well defined telnet port 23 by default. It can be any valid socket port number, i.e. a value between 1 and 65535. Depending on the operating system, it may not be possible to use a port number between 1 and 1024, as these values may have been reserved for security reasons. In such cases one needs to use a higher port number. Also, existing firewalls need to be configured to allow traffic to the defined `port`.

The attribute `keepClients` (values `.true` or `.false`) controls, whether connections to clients should be closed if the port number gets changed.

[8] In order to implement this interesting class it was necessary to devise a socket class first which would wrap ooRexx external socket functions into an ooRexx class, making it easy to create socket applications in ooRexx. The methods of the socket classes are listed in the appendix.

The attribute `maxLogsInQueue` determines the number of log messages that should be kept in a circular queue. If a new client connection is established, then the new client will get the latest `maxLogsInQueue` log messages sent right away.

3.4 The Layout 'log4r' Classes

Appenders can have a layout assigned to them, which will be used for formatting log messages. Figure 3.6 displays the `log4r` layout class hierarchy with the methods each layout class defines.

3.4.1 The “Layout” Class

The `Layout` class defines the attributes and methods that all specialized layout classes will possess.

The configurable attribute `contentType` accepts a string in the MIME (acronym for "Multimedia Internet Message Extensions", "Multipurpose Internet Mail Extensions") format, defaulting to the string `"text/plain"`. Depending on the layout the attributes `header` and `footer` allow to specify the string to be used for lead-in and lead-out text.

Simple formatting of a log message is implemented, listing the log messge number, followed by a colon and blank, depicting the log level, a blank surrounded dash, the string value of the log message, and the operating system dependent EOL ("end-of-line") characters. If a condition object is supplied, its content will be formatted as a string of key, colon, blank and value and appended, followed by another set of EOL characters.

3.4.2 The “SimpleLayout” Class

The `SimpleLayout` class simply specializes the `Layout` class to match the `'log4j'` SimpleLayout class. There are no specific methods defined, as all of the inherited methods already format log messages accordingly.

3.4.3 The “PatternLayout” Class

The `PatternLayout` class specializes the `Layout` class and allows the definition of a pattern, which gets used for formatting log messages. Depending on the pattern, all or only part of the information available in the passed in log message directory object may be used.

`log4r` adheres to the `'log4j'` conversion character encodings where possible, but also adds a few `log4r` specific options. A pattern is a sequence of characters which will be copied verbatimly into the created message string. If a percent character (`%`)

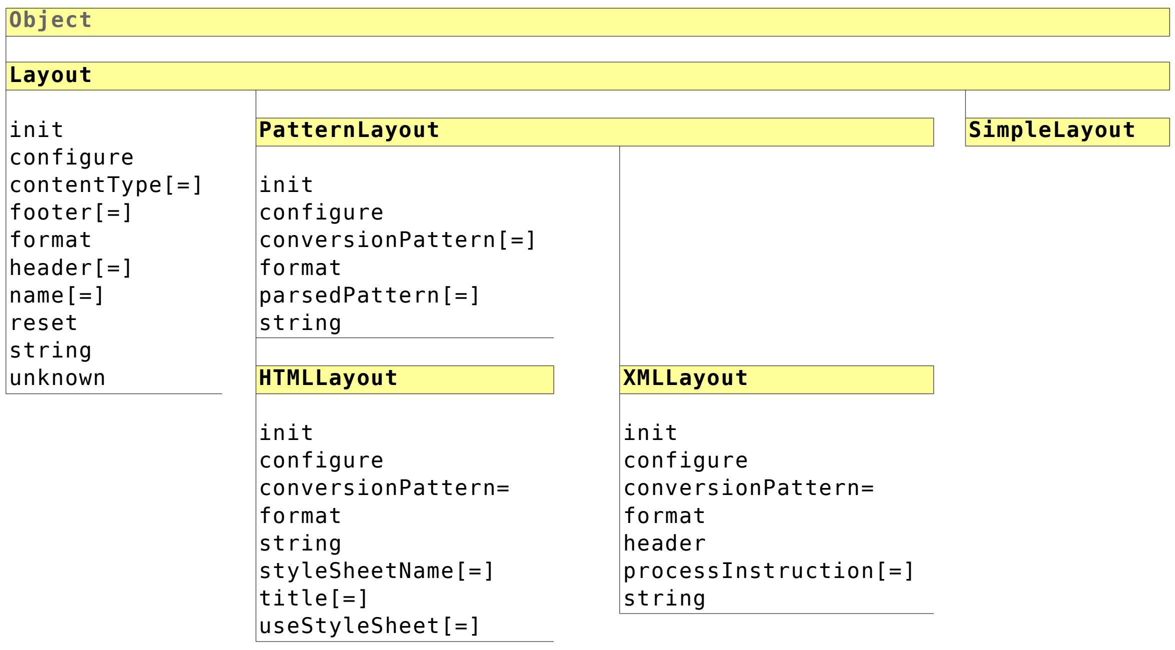


Figure 3.6: The Layout Class Hierarchy.

is encountered in the pattern string, special processing is applied. The percent character may be followed by one of the characters in figure 3.7 which may refer to an entry in the log message directory object, that got created by the logger and passed on to the layout `format` method via the appender. The meaning of each character is given in figure 3.7 in the comment field.

Between the escape character `%` and the pattern character of figure 3.7 there may be a number which determines the minimum width in the output string. Values which are shorter than the minimum width are right adjusted by default; wider values would not be truncated to the minimum width. If a value should be left or centrally adjusted within the minium width, then the escape character `%` needs to be immediately followed by a dash (`-`) or a carêt (`^`) character, respectively. Examples:

"`%10p`": minimum width 10, right adjust log level name,

"`%-10p`": minimum width 10, *left* adjust log level name,

"`%^10p`": minimum width 10, *center* log level name.

It is also possible to specify a maximum width, which is indicated by a dot followed by the maximum width number. Values that exceed the maximum width, will be truncated. Examples:

"`%.25m`": maximum width 25 characters, truncate message, if too long,

"`%10.25m`": minimum width 10, maximum width 25 characters, right adjust message, if shorter than minimum width,

"`%-25.25m`": minimum width 25, maximum width 25 characters, left adjust message, if shorter than minimum width, truncate to maximum width,

"`%^25.25m`": minimum width 25, maximum width 25 characters, center message, if shorter than minimum width, truncate to maximum width.

Character	***Comment***
a	**a**dditional, log4r only: refers to optional second argument, which can be a string or a directory object; in the case that the second argument to a log message is a condition object, one can submit the additional argument as a third argument to the log message.
c	**c**ategory: the logger name ("category" is the old `'log4j'` name for "logger"). If the name contains components delimited by dots, it is possible to indicate the number of components (*from the right!*) to be displayed by enclosing that number in curly brackets must follow immediatly the character, e.g.: `"%c{1}"` will extract the rightmost component from the logger's name and display it.
C	**C**lassName: n/a in log4r
d	**d**ateTime: the date and time the log message got created, default format: "YYYY-MM-DD hh:mm:ss.nnnnnn". The character may be immediately followed by a pair of curly brackets in which the first character may be "D" (date portion) or "T" (time portion). This allows to extract only the date or the time portion. If either the date or the time portion should be formatted in a a special way, then one letter may follow. This letter will be used to apply the ooRexx built-in-functions DATE() or TIME() for formatting and corresponds to the respective ooRexx letter. Finally, wherever DATE() allows to define a delimiter character (conversion character one of "ENOSU") with a conversion option, one can add another character representing that delimiter. Examples: "%d{DS-}" ... extract the date portion, convert it to a sorted date and use a dash as delimiter, "%d{TC}" ... extract the time portion, convert it to the U.S. "Civil" format.
f	**f**ieldName, log4r only: allows to propose a column title for the following log message information; e.g. used in .HTMLLayout. The desired name is enclosed in curly braces and must follow immediatly the character, e.g.: "%f{Elapsed Time}"
F	**f**ieldName: n/a in log4r
L	**L**ineNumber: n/a in log4r
n	**n**ewLine
N	Log**N**umber, log4r only: the logger's sequence number assigned to the log message
M	**M**ethodName: n/a in log4r
m	**m**essage
p	**p**riority, logLevel: the name of the log level ("priority" is the old `'log4j'` name for "log level")
R	elapsed time: formatted as a military (24) hour time, and if more than a day has passed since the timer got started, the number of days are given; hence the format is "d hh:mm:ss.nnnnnn"
r	elapsed time: formatted as seconds with microseconds, hence the format is: "s.nnnnnn"
t	**t**hreadName: n/a in log4r
%	print the **%** character verbatimly

Figure 3.7: Conversion Characters of the `PatternLayout` Class.

3.4.4 The “HTMLLayout” Class

The `HTMLLayout` class specializes the `PatternLayout` class and renders log messages with HTML encodings. The `conversionPattern` determines what kind of information gets created. If a "`%f{someFieldColumnHeading}`" precedes a conversion character, then the bracketed string serves as the column heading text in the HTML table this layout will use to format the log messages.

If the attribute `useStyleSheet` is set to `.true`, then a style sheet will be used for displaying the HTML table. In that case, if the attribute `styleSheetName` is blank an internal style element will be created in the HTML head element, otherwise a link to the external style sheet with the stored file name will be inserted.

3.4.5 The “XMLLayout” Class

The `XMLLayout` class specializes the `PatternLayout` class and renders log messages with XML encodings. The `conversionPattern` determines what kind of information gets created. If a "`%f{someFieldColumnHeading}`" precedes a conversion character, then the bracketed string serves as the XML element name.

If the attribute `processInstruction` contains a non-empty string, then it is inserted into the XML header.

The default XML encoding is self documenting.

3.5 The Filter 'log4r' Classes

Appenders possess a `filterQueue` attribute, which may be used to store any number of filters. If there are filters available, the appender presents the log message directory object to each filter and processes such a log message only, if it does not get denied in the process.

A filter may be neutral about a log message, which will cause the log message to be presented to the next filter in the queue. If there are no more filters in the queue, the log message gets processed.

Some filters may accept a log message, in which case the log message is not presented to possibly remaining filters in the queue, but is rather processed right away.

3.5.1 The “Filter” Class

The `Filter` class defines the attributes and methods that all specialized filter classes will possess.

The configurable attribute `acceptOnMatch` can be set to `.true` or `.false`. However, it depends on the semantics of the particular filter type what these values mean, whether the decision of the filter would be "accept", "deny" or "neutral" as a result.

Figure 3.8 on p. 25 depicts the `log4r` filter class hierarchy.

3.5.2 The “DateRangeFilter” Class

The `DateRangeFilter` class specializes the `Filter` class and allows the definition of a date range, assigning date values (strings formatted like sorted dates, i.e. YYYYMMDD) to the attributes `dateMin` (default: "`00010101`") and `dateMax` (default: "`99991231`").

If there is an error in the filter's definition, i.e. `dateMax<dateMin`, then the filter will return the decision *"neutral"*.

This filter takes the following decisions, if the log message date falls within the given date range:

- if `acceptOnMatch` is set to `.true`, it returns *"neutral"*,
- if `accetpOnMatch` is set to `.false`, it returns *"deny"*.

This filter takes the following decisions, if the log message date does *not* fall within the given date range:

- if `acceptOnMatch` is set to `.true`, it returns *"deny"*,
- if `accetpOnMatch` is set to `.false`, it returns *"neutral"*.

3.5.3 The “DenyAllFilter” Class

The `DenyAllFilter` class specializes the `Filter` class and always returns the decision *"deny"*. Sometimes it may be useful to place such a filter at the end of the filter queue, in case the programmer wants a log message not to be processed, if the filters in the queue only return the decision *"neutral"*.

3.5.4 The “LevelMatchFilter” Class

The `LevelMatchFilter` class specializes the `Filter` class and allows to define a log level with the attribute `logLevelToMatch` to test the log message against.

If a log message does not possess the same log level as the attribute `logLevelToMatch`, then the decision *"neutral"* is returned.

Otherwise (log level of message matches this filter's log level) the following decisions are taken by this filter:

- if `acceptOnMatch` is set to `.true`, it returns *"accept"*,
- if `accetpOnMatch` is set to `.false`, it returns *"deny"*.

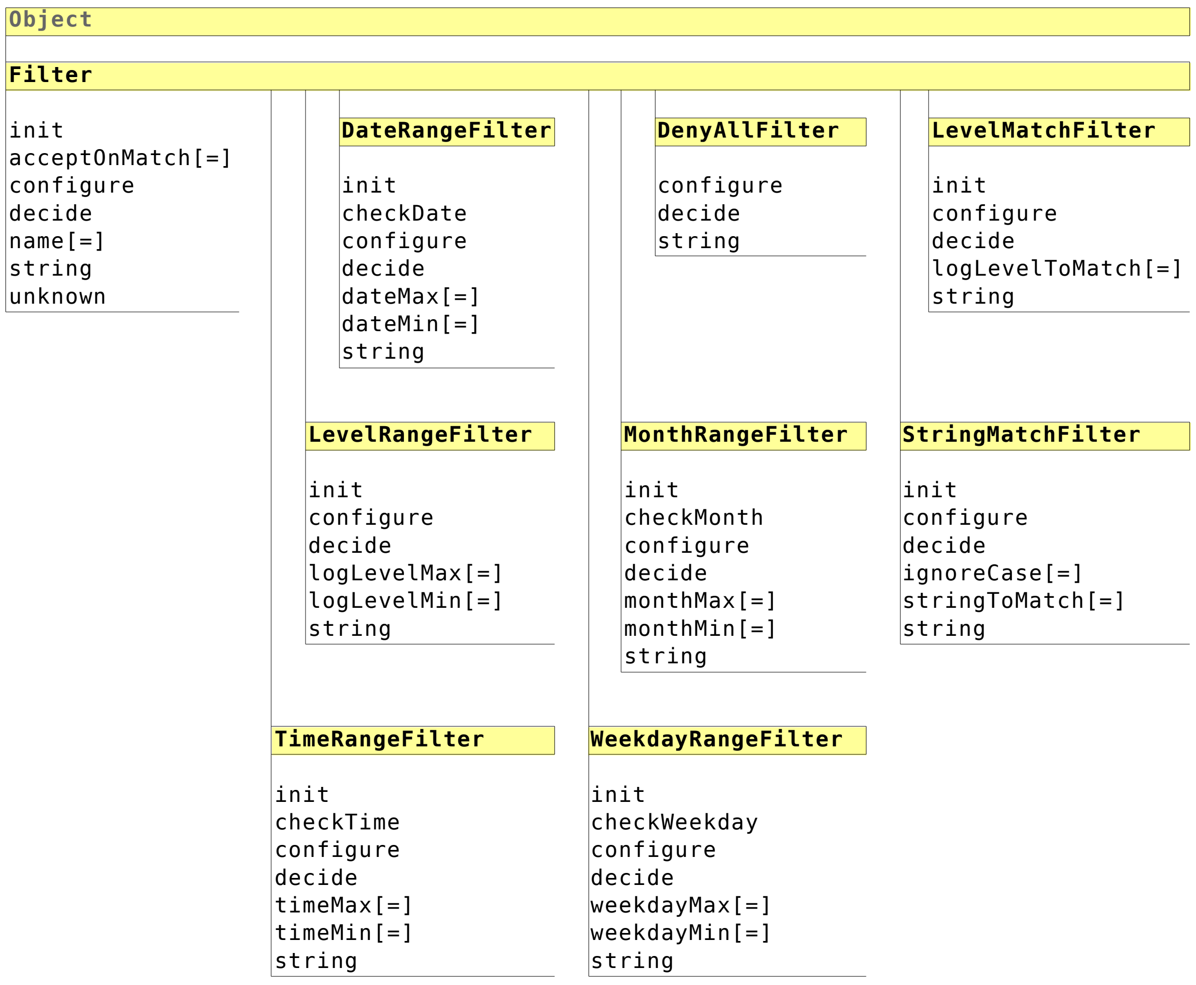


Figure 3.8: The Filter Class Hierarchy.

3.5.5 The “LevelRangeFilter” Class

The `LevelRangeFilter` class specializes the `Filter` class and allows the definition of a log level range, assigning log level values to the attributes `logLevelMin` (default: "`LOWEST`") and `logLevelMax` (default: "`HIGHEST`").

This filter takes the following decisions, if the log message's log level falls within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"accept"*,
- if `accetpOnMatch` is set to `.false`, it returns *"neutral"*.

This filter takes the following decision, if the log message's log level does *not* fall within the given range: it returns *"deny"*.

3.5.6 The “MonthRangeFilter” Class

The `MonthRangeFilter` class specializes the `Filter` class and allows the definition of a month range, assigning month values (either the numbers "`1`" through "`12`", or the English month names "`January`" through "`December`") to the attributes `monthMin` (default: "`1`", January) and `monthMax` (default: "`12`", December).

Note that it is possible to have `monthMax<monthMin`, which means that the month range spans a year. E.g. defining `monthMin=11`, and `monthMax=1` means a month range from `"November"` through `"January"`.

This filter takes the following decisions, if the log message's month falls within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"neutral"*,
- if `accetpOnMatch` is set to `.false`, it returns *"deny"*.

This filter takes the following decisions, if the log message's month does *not* fall within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"deny"*,
- if `accetpOnMatch` is set to `.false`, it returns *"neutral"*.

3.5.7 The “StringMatchFilter” Class

The `StringMatchFilter` class specializes the `Filter` class and allows to define a search string stored in the attribute `stringToMatch` to test the log message against. If the attribute `ignoreCase` (default value: `.false`) is set to `.true`, the comparison is carried out caselessly.

If the search string of the attribute `stringToMatch` or the message text is the empty string, or the message text is `.nil`, then the decision *"neutral"* is returned.

Otherwise (log level of message matches this filter's log level) the following decisions are taken by this filter:

- if `acceptOnMatch` is set to `.true`, it returns *"accept"*,
- if `accetpOnMatch` is set to `.false`, it returns *"deny"*.

If the search string is not found in the message text, then *"neutral"* gets returned.

3.5.8 The “TimeRangeFilter” Class

The `TimeRangeFilter` class specializes the `Filter` class and allows the definition of a time of day range, assigning time values (in the form "`mm:hh`") to the attributes `timeMin` (default: "`00:00`") and `timeMax` (default: "`23:59`"). The comparisons do not take seconds and microseconds into account..

Note that it is possible to have `timeMax<timeMin`, which means that the time of day range spans a day. E.g. defining `timeMin="22:00"`, and `timeMax="01:00"` means a time of day range range from `"22:00"` evening through `"01:00"` in the morning of the next day.

This filter takes the following decisions, if the log message's time of day falls within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"neutral"*,
- if `accetpOnMatch` is set to `.false`, it returns *"deny"*.

This filter takes the following decisions, if the log message's time of day does *not* fall within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"deny"*,
- if `accetpOnMatch` is set to `.false`, it returns *"neutral"*.

3.5.9 The “WeekdayRangeFilter” Class

The `WeekdayRangeFilter` class specializes the `Filter` class and allows the definition of a weekday range, assigning weekday values (either the numbers "`1`", representing Monday, through "`7`", representing Sunday, or the English weekday names "`Monday`" through "`Sunday`") to the attributes `weekdayMin` (default: "`1`", Monday) and `weekdayMax` (default: "`7`", Sunday).

Note that it is possible to have `weekdayMax<weekdayMin`, which means that the weekday range spans a week. E.g. defining `weekdayMin="6"`, and `weekdayMax="1"` weekday range range from `"Saturday"` through `"Monday"`.

This filter takes the following decisions, if the log message's weekday falls within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"neutral"*,
- if `accetpOnMatch` is set to `.false`, it returns *"deny"*.

This filter takes the following decisions, if the log message's time of day does *not* fall within the given range:

- if `acceptOnMatch` is set to `.true`, it returns *"deny"*,
- if `accetpOnMatch` is set to `.false`, it returns *"neutral"*.

4 Configuring 'log4r'

The log4r framework implements the ability to configure all loggers in a plain text file, a so called "property file". Such a file contains line-delimited entries in the form of pairs of key and their values delimited by an assignment character (=). Following the Rexx philosophy the key is case independent (keys will be uppercased), whereas the case and the leading and trailing blanks in the value are preserved, as that may be significant (e.g. for StringMatchFilter).

Empty lines, and lines that start with a hash (#), a semi-colon (;), an exclamation mark (!) or two consecutive dashes (--) are ignored.

When the framework starts up it first checks whether the environment symbol ".log4r.config.configFile" is defined and denotes an existing file. If so that file is taken as the property file containing the configuration to be used by the log4r framework. Otherwise the following directories are searched first for a file named "log4r.properties", and if not found for a file named "simplelog4r.properties": the current directory, the home directory of the log4r framework, and then all directories listed in the PATH environment variable. The first hit will be taken and that property file will be read and used for configuring the runtime. If no property file can be found, then a default configuration is created, setting the environment symbol "log4r.config.LoggerFactoryClassName" to "NoOpLog", which will make sure that no log messages sent by applications cause an error, but also, that they will not be processed.

The default configuration settings are depicted in figure 4.1.

```
log4r.config.asyncMode               =.false
log4r.config.configFile              =.nil
log4r.config.configFileWatchInterval = 0
log4r.config.disable                 = LOWEST
log4r.config.disableOverride         =.false
log4r.config.LoggerFactoryClassName  = NoOpLog
log4r.config.LogLog.debug            =.false
log4r.config.LogLog.quietMode        =.false
log4r.config.version                 =101.20070423
```

Figure 4.1: The log4r Default Configuration.

If an option accepts a logical (Boolean) value, then one may use the values 0, "false" or ".false" to represent the truth value "*false*" and the values 1, "true" or ".true" for representating the truth value "*true*".

The configuration entries from a property file are stored as a log4r.Properties object with the LogManager class attribute configuration.

As a general rule unknown entries (unknown keys) in the property file are simply ignored and do not cause an error to be raised. This way it becomes possible to use a single property file for 'log4r' and for 'log4j' as the lead-in string for the keys will be different for both frameworks.

4.1 Global Configuration Settings

Configuration settings that affect the overall execution of the framework start with the lead-in string "log4r.config." and are stored in the ooRexx .local environment. As such these entries can be retrieved using the environment symbol notation, i.e. prepending the property name with a dot.

Figure 4.2 lists the overall configuration options with a brief description of their purpose. It also documents the default configuration settings for loggers of type "SimpleLog".

Key	***Brief Description***
log4r.config.asyncMode	If set to .true the logger processes the log message asynchroneously. Default value: .false.
log4r.config.configFile	If set, denotes the name of a configuration file.
log4r.config.configFileWatchInterval	If set to a value greater than 0, then this is the interval time in seconds to check whether the configuration file has been changed, and if so re-read it and apply the changes to the running system.
log4r.config.disable	Determines the threshold log level in order for all the loggers to process log messages. One of LOWEST, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, HIGHEST. Default value: lowest.
log4r.config.disableOverride	If set to .true all log messages will be processed by the loggers, independently of the setting log4r.config.disable. Default value: .false.
log4r.config.LoggerFactoryClassName	Set to the name of the logger class that should be used to create loggers, one of Log, SimpleLog and NoOpLog. Default value: NoOpLog.
log4r.config.LogLog.debug	If set to .true the debug log messages from the log4r framework classes are displayed. Default value: .false.
log4r.config.LogLog.quietMode	If set to .true the warn and error in addition to the debug log messages from the log4r framework classes are *not* displayed. Default value: .false.
Default Settings for "SimpleLog" Loggers	
log4r.config.simplelog.defaultLogLevel	Determines the default threshold log level for SimpleLog loggers. One of LOWEST, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, HIGHEST. Default value: info.
log4r.config.simplelog.showDateTime	If set to .true then the date of the log message is displayed. Default value: .false.
log4r.config.simplelog.showLoggerName	If set to .true then the logger's name is displayed. Default value: .false.
log4r.config.simplelog.showShortName	If log4r.config.simplelog.showLoggerName is set to .true and this option is set to .true then the last component of the logger's name is displayed. Default value: .false.

Figure 4.2: Overall Configuration Keys with a Brief Description.

4.2 Logger Configuration Settings

Logger configuration settings start with the lead-in string "log4r.logger." If the remaining string does not end in ".additivity" (for loggers of type "Log") or one (for loggers of type "SimpleLog") of ".showDateTime", ".showLoggerName", ".showShortName", then it is taken as the name of the logger. The name of a logger may contain dots, which then separate the "components" of the logger's name.

The value part is a comma separated list of words, where the first word determines the threshold log level, followed by one or more words which are the names of defined appenders to which the logger should append the received log messages to.

Key	***Brief Description***
log4r.logger.NAME	Defines the NAME of the logger, which may contain dots. The value is a list of comma-separated words, where the first word denotes the threshold log level and the following words are the names of defined appenders to which a log message should be sent to.
log4r.logger.NAME.additivity	Optional, if set to .false the logger named NAME will not send the log message to its parent logger for further processing. Default value: .true.

Figure 4.3: Configuration Keys for Loggers of Type 'Log" with a Brief Description.

Figure 4.4 gives a few examples of defining and configuring loggers. Depending on the setting of the environment symbol .log4r.config.LoggerFactoryClassName either the class "Log", "SimpleLog" or "NoOpLog" is used to create the logger instances. Figure 4.3 lists the configuration options for defining loggers of type "Log", figure 4.5 lists the configuration options for defining loggers of type "SimpleLog".

If the loggers are created using the "Log" class, the listed appenders are honored. If the names of appenders are given, that do not exist, then an appropriate error log

```
-- define and configure a logger named "this.is.logger.one"
log4r.logger.this.is.logger.one=debug, appender1

-- define and configure a logger named "this.is", a parent logger to "this.is.logger.one"
log4r.logger.this.is=warn, appender3
-- do not forward log message to parent logger (would be "rootLogger" in this case)
log4r.looger.this.is.additivity=false

-- configure the "rootLogger" (will be created by the framework)
log4r.logger.rootLogger=error, appender2

-- define and configure a logger named "some.other.logger"
log4r.logger.some.other.logger=trace, appender1, appender2, appender3

-- define and configure a logger, supply settings, if the 'SimpleLog' class is used
log4r.logger.yet.another.logger = debug
log4r.logger.yet.another.logger.showDateTime=true
log4r.logger.yet.another.logger.showFileName=true
log4r.logger.yet.another.logger.showShortName=true
```

Figure 4.4: Examples for Defining and Configuring Loggers of Type 'Log".

message is created to warn the user and the appender will be ignored. If the appender queue is empty and the attribute `additivity` is set to its default value `.true`, then log messages are forwarded to the logger's parent for further processing. If no parent with the same stem exists then the framework's root logger named `rootLogger` is taken as the parent logger.

Key	***Brief Description***
`log4r.logger.NAME`	Defines the `NAME` of the logger, which may contain dots. The value is a list of comma-separated words, where the first word denotes the threshold log level and the following words are the names of appenders to which a log message should be sent to.
`log4r.logger.NAME.showDateTime`	Optional, if set to `.true` then the date of the log message is displayed. Default value: `.false`.
`log4r.logger.NAME.showLoggerName`	Optional, if set to `.true` then the logger's name is displayed. Default value: `.false`.
`log4r.logger.NAME.showShortName`	Optional, if `log4r.logger.NAME.showLoggerName` is set to `.true` and this option is set to `.true` then the last component of the logger's name is displayed. Default value: `.false`.

Figure 4.5: Configuration Keys for Loggers of Type '`SimpleLog`" with a Brief Description.

4.3 Appender Configuration Settings

Appender configuration settings start with the string "`log4r.appender.`". If the remaining string does not contain a dot, then it is taken as the `name` of the appender. The value part will denote the name of the appender class that should be used to create the appender instance.

Figure 4.6 lists all the configuration options that all appenders possess. If the names of filters are given, that do not exist, then an appropriate error log message is created to warn the user and the filter will be ignored.

Key	***Brief Description***
`log4r.appender.NAME`	Defines the `NAME` of the appender, which must not contain any dots. The value is the name of the appender class that should be used to create the appender instance.
`log4r.appender.NAME.filter`	Optional. The value is a list of comma-separated words, where each word denotes the name of a defined filter to be used to determine whether the appender should process the log message.
`log4r.appender.NAME.layout`	Optional. Names a defined layout to be used for formatting the log message. Default value: `.nil`.
`log4r.appender.NAME.requiresLayout`	Optional, if set to `.true` then a layout object is required for this appender. Default value: `.false`.
`log4r.appender.NAME.threshold`	Optional. Determines the default threshold log level for the appender. One of `LOWEST/ALL`, `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `HIGHEST/OFF/NONE`. Default value: `lowest`.

Figure 4.6: Configuration Keys for All Appenders with a Brief Description.

4.3.1 ConsoleAppender Configuration Settings

A console appender has in addition to the settings in figure 4.6 the settings that are documented and briefly described in figure 4.7.

Key	***Brief Description***
`log4r.appender.NAME.follow`	Optional, if set to `.false` then an attempt to change the `target` attribute is not followed (is ignored). Default value: `.true`.
`log4r.appender.NAME.immediateFlush`	Optional, if set to `.false` then the output of the log message *may* be buffered. Default value: `.true` (flush the buffer after writing the log message).
`log4r.appender.NAME.target`	Optional, determines which standard file is to be used for writing the log messages to: `stdout` or `stderr` . Default value: `stderr`.

Figure 4.7: Configuration Keys for a ConsoleAppender with a Brief Description.

4.3.2 FileAppender Configuration Settings

A file appender has in addition to the settings in figure 4.6 the settings that are documented and briefly described in figure 4.8.

Key	***Brief Description***
`log4r.appender.NAME.append`	Optional, if set to `.false` then the file's content gets truncated before starting to write the log messages to the file. Default value: `.true` (append the log messages to the end of the file).
`log4r.appender.NAME.bufferedIO`	Optional, if set to `.true` then the output of the log message *may* be buffered. Default value: `.false` (flush the buffer after writing the log message).
`log4r.appender.NAME.fileName`	Optional, determines the filename, which may be fully qualified. Default value: `FileAppenderDefault.log`.

Figure 4.8: Configuration Keys for a FileAppender with a Brief Description.

4.3.2.1 DailyRollingFileAppender Configuration Settings

A daily rolling file appender allows automatic switching of the used log files on predefined date/time intervals. It extends (specializes) the FileAppender above and adds additional configuration settings as depicted in figure 4.9.

Key	***Brief Description***
`log4r.appender.NAME.rollType`	Optional. Allows setting the daily rolling type, which may be one of: `MINUTE`, `HOUR`, `HALF_DAY`, `DAY`, `WEEK`, `MONTH`. Default value: `DAY`.

Figure 4.9: Configuration Keys for a DailyRollingFileAppender with a Brief Description.

4.3.2.2 RollingFileAppender Configuration Settings

A rolling file appender allows automatic switching of the used log file, if a predefined file size gets exceeded. It extends (specializes) the FileAppender above and adds additional configuration settings as depicted in figure 4.10 below.

Key	*Brief Description*
`log4r.appender.NAME.maxBackupIndex`	Optional. A number which determines how many different backup log files should be kept. If set to `0`, then no backup file is created and the log file will be deleted when rolling over. Default value: `1`.
`log4r.appender.NAME.maxFileSize`	Optional. Indicates the maximum number of bytes in the file, before a roll over occurs. The number can be followed by one of `kb`, `mb`, `gb`, `tb`. Default value: `10mb`.

Figure 4.10: Configuration Keys for a RollingFileAppender with a Brief Description.

4.3.3 TelnetAppender Configuration Settings

A telnet appender has in addition to the settings in figure 4.6 the settings that are documented and briefly described in figure 4.11 below.

Key	*Brief Description*
`log4r.appender.NAME.keepClients`	Optional, if set to `.true` then changing the value of the attribute `port` will not cause established client connections to be closed. Default value: `.false`.
`log4r.appender.NAME.maxLogsInQueue`	Optional. A whole number determining the number of log messages to be kept in a buffer. If a new client connects the last `maxLogsInQueue` log messages are sent to it Default value: `25`.
`log4r.appender.NAME.port`	Optional, determines the TCP/IP socket port to listen for clients to serve. Default value: `25` ("well known Telnet port").

Figure 4.11: Configuration Keys for a TelnetAppender with a Brief Description.

4.4 Layout Configuration Settings

Layout configuration settings start with the string "`log4r.layout.`".

If the remaining string does not contain a dot, then it is taken as the name of the layout. The value part will denote the name of the layout class that should be used to create the layout instance.

Figure 4.12 lists all the configuration options that all layouts possess.

Key	*Brief Description*
`log4r.layout.NAME`	Defines the `NAME` of the layout, which must not contain any dots. The value is the name of the layout class that should be used to create the layout instance.
`log4r.layout.NAME.contentType`	Optional. The content type in MIME format. Default value: `text/plain`.
`log4r.layout.NAME.footer`	Optional. A string that should be used as a footer. Default value: empty string.
`log4r.layout.NAME.header`	Optional. A string that should be used as a footer. Default value: empty string.

Figure 4.12: Configuration Keys for All Layouts with a Brief Description.

4.4.1 PatternLayout Configuration Settings

A pattern layout has in addition to the settings in figure 4.12 the settings that are documented and briefly described in figure 4.13.

Key	*Brief Description*
`log4r.layout.NAME.conversionPattern`	Optional, a string containing a conversion pattern. Default value: `%5N: %r [%c] %-5p - %m%n`.

Figure 4.13: Configuration Keys for a PatternLayout with a Brief Description.

4.4.1.1 HTMLLayout Configuration Settings

A HTML layout allows formatting log messages as HTML marked up text, following the `conversionPattern` attribute, which is set to the following default value:

```
%f{LogNr}%N %f{DateTime}%d %f{ElapsedTime}%r %f{Logger}[%c] %f{LogLevel} %^p %f{Message}%-m
```

It extends (specializes) the PatternLayout above and adds additional configuration settings as depicted in figure 4.14.

Key	*Brief Description*
`log4r.layout.NAME.styleSheetName`	Optional string which denotes the name of a cascading style sheet file. Default value: empty string.
`log4r.layout.NAME.useStyleSheet`	Optional, if set to `.false` then no style information is generated. If set to `.true`, and a `styleSheetName` is given, a link to that CSS file is inserted, otherwise a style element is created for the HTML head element. Default value: `.true`.

Figure 4.14: Configuration Keys for a HTMLLayout with a Brief Description.

4.4.1.2 XMLLayout Configuration Settings

A XML layout allows formatting log messages as XML marked up text, following the `conversionPattern` attribute, which is set to the following default value:

```
#%N: %d %r [%c] %-5p - %m%n
```

It extends (specializes) the PatternLayout above and adds additional configuration settings as depicted in figure 4.15.

Key	*Brief Description*
`log4r.layout.NAME.processInstruction`	Optional string which denotes one or more fully formed PI (process instructions) which will be inserted in the XML header. Default value: empty string.

Figure 4.15: Configuration Keys for a XMLLayout with a Brief Description.

4.5 Filter Configuration Settings

Filter configuration settings start with the string "`log4r.filter.`". If the remaining string does not contain a dot, then it is taken as the `name` of the filter. The value part will denote the name of the filter class that should be used to create the filter instance. Figure 4.16 lists the configuration options that all filters possess.

Key	***Brief Description***
`log4r.filter.NAME`	Defines the `NAME` of the filter, which must not contain any dots. The value is the name of the filter class that should be used to create the filter instance.
`log4r.filter.NAME.acceptOnMatch`	Optional logical value. Default value: `.true`.

Figure 4.16: Configuration Keys for All Filters with a Brief Description.

4.5.1 DateRangeFilter Configuration Settings

A date range filter has in addition to the settings in figure 4.16 the settings that are documented and briefly described in figure 4.17 below.

Key	***Brief Description***
`log4r.filter.NAME.dateMin`	Optional, a sorted (formatted as "YYYYMMDD") date. Default value: `00010101`.
`log4r.filter.NAME.dateMax`	Optional, a sorted (formatted as "YYYYMMDD") date. Default value: `99991231`.

Figure 4.17: Configuration Keys for a DateRangeFilter with a Brief Description.

4.5.2 LevelMatchFilter Configuration Settings

A level match filter has in addition to the settings in figure 4.16 the setting that is documented and briefly described in figure 4.18 below.

Key	***Brief Description***
`log4r.filter.NAME.logLevelToMatch`	Optional, one of `LOWEST`/`ALL`, `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `HIGHEST`/`OFF`/`NONE`. Default value: `lowest`.

Figure 4.18: Configuration Keys for a LevelMatchFilter with a Brief Description.

4.5.3 LevelRangeFilter Configuration Settings

A level range filter has in addition to the settings in figure 4.16 the settings that are documented and briefly described in figure 4.19 below.

Key	***Brief Description***
`log4r.filter.NAME.logLevelMin`	Optional, one of `LOWEST`/`ALL`, `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `HIGHEST`/`OFF`/`NONE`. Default value: `lowest`.
`log4r.filter.NAME.logLevelMax`	Optional, one of `LOWEST`/`ALL`, `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `HIGHEST`/`OFF`/`NONE`. Default value: `highest`.

Figure 4.19: Configuration Keys for a LevelMatchFilter with a Brief Description.

4.5.4 MonthRangeFilter Configuration Settings

A month range filter has in addition to the settings in figure 4.16 the settings that are documented and briefly described in figure 4.20 below.

Key	***Brief Description***
`log4r.filter.NAME.monthMin`	Optional, either the number (e.g. "`1`" for January) of the month or the English name of the month (e.g. `January`). Default value: `1`.
`log4r.filter.NAME.monthMax`	Optional, either the number (e.g. "`12`" for December) of the month or the English name of the month (e.g.`December`). Default value: `12`.

Figure 4.20: Configuration Keys for a MonthRangeFilter with a Brief Description.

4.5.5 StringMatchFilter Configuration Settings

A string match filter has in addition to the settings in figure 4.16 the settings that are documented and briefly described in figure 4.21 below.

Key	***Brief Description***
`log4r.filter.NAME.ignoreCase`	Optional, if set to `.true` then the `stringToMatch` is caselessly compared with the log message text. Default value: `.false`.
`log4r.filter.NAME.stringToMatch`	Optionally a string. Default value: empty string.

Figure 4.21: Configuration Keys for a StringMatchFilter with a Brief Description.

4.5.6 TimeRangeFilter Configuration Settings

A time range filter has in addition to the settings in figure 4.16 the settings that are documented and briefly described in figure 4.22 below.

Key	***Brief Description***
`log4r.filter.NAME.timeMin`	Optional, a time of day in the form "mm:hh". Default value: `00:00`.
`log4r.filter.NAME.timeMax`	Optional, a time of day in the form "mm:hh". Default value: `23:59`.

Figure 4.22: Configuration Keys for a TimeRangeFilter with a Brief Description.

4.5.7 WeekdayRangeFilter Configuration Settings

A weekday range filter has in addition to the settings in figure 4.16 the settings that are documented and briefly described in figure 4.23 below.

Key	***Brief Description***
`log4r.filter.NAME.weekdayMin`	Optional, either the number (e.g. "`1`") for the day of the week or its English name (e.g. `Monday`). Default value: `1`.
`log4r.filter.NAME.weekdayMax`	Optional, either the number (e.g. "`7`") for the day of the week or its English name (e.g. `Sunday`). Default value: `7`.

Figure 4.23: Configuration Keys for a WeekdayRangeFilter with a Brief Description.

5 Summary and Outlook

This article introduced the new log4r framework that got created for the 18th International Rexx Symposium in 2007. The framework is closely modelled after Apache's log4j, but takes advantage of some of ooRexx features, like the "environment" which serves as a globally available directory to ooRexx programs for storing configuration data and the log4r classes. Therefore e.g. the LogManager class object can be directly referenced via its environment symbol .LogManager.

The names of configuration keys for property files are systemized, in that global configuration options must start with the string "log4r.configuration." Layouts and filters are named and can therefore be re-used for different appenders.

Taking advantage of log4r is easy, one merely nees to write a statement to "call load_log4r.rex". It is possible to set global configuration settings for the log4r framework, before calling it, by defining the global configuration settings as entries in the .local ooRexx environment.

After the initialization of the log4r framework, one is able to retrieve a logger at any time by merely asking the .LogManager for it by name (sending it the message getLogger('name.of.logger'). If a logger by that name does not exist yet, it will be created with the default values and stored in the .LogManager directory of loggers. The returned logger immediately allows for sending log messages at different levels to it, depending on the message one sends to the logger (trace, debug, info, warn, error, fatal).

The entire framework can be easily extended, such as to add new appenders (e.g. mail or instant messaging appenders which may send failure log messages), layouts and filters. Its source is open and freely available and should serve as a documentation for the APIs, given the comments supplied in the source code. As log4j was used to model log4r it should be possible to use the log4j documentation, tutorial and articles to learn more how this style of logging can be applied and taken advantage from. Also, one can draw many ideas from the ongoing development of log4j which could be applied to log4r, if they look promising or benefitial.

A last remark: the ooRexxUnit framework [Flat06] was heavily used to create unit tests for the methods of the classes and routines of the log4r framework.

6 References

[Cow90] Cowlishaw, M.F.: "The REXX Language", Prentice-Hall (Second edition), 1990.

[Flat06] Flatscher R.G.: "ooRexxUnit: A JUnit Compliant Testing Framework for ooRexx Programs", in: Proceedings of the "The 2006 Interational Rexx Symposium", Austin, Texas, U.S.A. April 9th – April 13th 2006.

[Fos05] Fosdick H.: "Rexx Programmer's Reference", John Wiley & Sons, ISBN: 0-7645-7996-7, URL (as of 2007-04-22): `http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764579967.html`

[ooRexx] URL (as of 2007-04-22): `http://www.ooRexx.org`

[Rexx] URL (as of 2007-04-22): `http://www.Rexx.org`

[RexxInfo] URL (as of 2007-04-22): `http://www.RexxInfo.org`

[RexxLA] URL (as of 2007-04-22): `http://www.RexxLA.org`

[VeTrUr] Veneskey G.L., Trosky W., Urbaniak J.J.: "Object Rexx by Example", Aviar. URL (as of 2007-04-22): `http://www.oops-web.com/orxbyex/`

[W3G02] Gülcü C.: "Short introduction to log4j", URL (as of 2007-04-22): `http://logging.apache.org/log4j/docs/manual.html`

[W3G04] Homepage of Gülcü C.: "The complete log4j manual", PDF book, last updated: 2004-12-12, URL (as of 2007-04-22): `https://www.qos.ch/shop/products/log4j/log4j-Manual.jsp`

[W3L4J] Homepage of Apache's 'log4j' Framework, URL (as of 2007-04-22): `http://logging.apache.org/log4j/docs/`

[W3L4R] Homepage for the 'log4r' Framework, URL (as of 2007-04-22): `http://wi.wu-wien.ac.at/rgf/rexx/orx18/log4r/`

[W3LOG] Homepage of Apache's "Logging Services", URL (as of 2007-04-22): `http://logging.apache.org/`

[W3M05] Mills A.J.S.: "Log4J", URL (as of 2007-04-22): `http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/log4j/log4j.html`

[W3OORC] Code repository for ooRexx (as of 2007-04-22): `http://oorexx.cvs.sourceforge.net/oorexx/`

[W3ORUC] Code repository for ooRexxUnit (as of 2007-04-22): `http://oorexx.svn.sourceforge.net/viewvc/oorexx/test/`

[W3S01] Vipan S.: "Don't Use System.out.println! Use Log4j", URL (as of 2007-04-22): `http://www.vipan.com/htdocs/log4jhelp.html`

7 Appendix A. Example "log4r.properties"

The log4r framework is distributed with an example "log4r.properties" file which originally looked like the one of this appendix.

```
; Sample "log4r.properties" file, Rony G. Flatscher, 2007-04-22
;
; a simple 'log4r' sample configuration file
;
; - entries are strings in the form: key=value
;   attention: case of key is irrelevant (as in Rexx everything gets translated to uppercase)
;
; - empty lines or lines starting with ';' or '#' or '!' or '--' are ignored
;
; - these are the defined logLevels: TRACE < DEBUG < INFO < WARN < ERROR < FATAL
;   just send messages of one of those logLevel names to a logger
;
; - anytime you need a logger, get it by querying the ".LogManager", e.g.:
;
;      aLogger=.LogManager~getLogger("rootLogger") -- get the rootLogger
;
;   then send a log message, e.g.:
;
;      aLogger~trace("This is a trace message")
;      aLogger~debug("This is a debug message")
;      aLogger~info("This is an info message")
;      aLogger~warn("A condition got raised!", condition('O'))
;      aLogger~error("This is an error message")
;      aLogger~fatal("This is a fatal (error) message")


# =======================================================================================
; turn on 'log4r' framework debugging:
# log4r.config.LogLog.debug=1
;
; turn even logging of framework warnings and errors off:
# log4r.config.LogLog.quietMode=1
;
# =======================================================================================


# ====================== define and configure loggers
# ---------------------- logger named "rootLogger" (this logger is always available)
;
-- name of the following logger: "rootLogger"
; send logs at level "DEBUG" or higher to the appender named "DEST_APP1"
log4r.logger.rootLogger = debug, dest_app1


# ---------------------- logger named "rgf.sockets" (meant to serve 'rgf.sockets.cls' module)
;
; send logs at level "TRACE" or higher to the appender named "RGF_APP1" and "RGF_APP2"
-- name of the following logger: "RGF.SOCKETS"
log4r.logger.RGF.SOCKETS           = debug, rgf_app1, rgf_app2
;
; if 'additivity' is .true (default), then log messages are sent to the parent logger
log4r.logger.rgf.sockets.additivity=false

; ... continued on next page ...
```

```
# ====================== define and configure appenders
; ---------------------- appender named "DEST_APP1"
log4r.appender.DEST_APP1               =ConsoleAppender
log4r.appender.dest_app1.ImmediateFlush=true
log4r.appender.dest_app1.layout        =pat_layout1
log4r.appender.dest_app1.Target        =stderr


; ---------------------- appender named "RGF_APP1"
log4r.appender.RGF_APP1                =ConsoleAppender
log4r.appender.rgf_app1.filter         =weekDays,workingHours
log4r.appender.rgf_app1.ImmediateFlush =1
log4r.appender.rgf_app1.layout         =pat_layout1
log4r.appender.rgf_app1.Target         =.error


; ---------------------- appender named "RGF_APP2"
log4r.appender.RGF_APP2                = FileAppender
log4r.appender.rgf_app2.filter         = weekDays,workingHours
log4r.appender.rgf_app2.layout         = html_layout1
log4r.appender.rgf_app2.fileName       = rgf_app2_appender.html
log4r.appender.rgf_app2.append         = false
log4r.appender.rgf_app2.bufferedIO     = false

; ---------------------- appender named "RGF_APP3"
log4r.appender.RGF_APP3                = TelnetAppender
log4r.appender.rgf_app3.layout         = pat_layout1
log4r.appender.rgf_app3.maxLogsInQueue = 50
;
; uncomment to ignore all logs up to, but not including level 'WARN'
; log4r.appender.rgf_app3.Threshold      = warn


# ====================== define and configure layouts
; ---------------------- layout named "PAT_LAYOUT1"
log4r.layout.PAT_LAYOUT1           =PatternLayout
log4r.layout.pat_layout1.conversionPattern=%5N: %r [%c] %-5p - %m%n

; ---------------------- layout named "HTML_LAYOUT1"
log4r.layout.HTML_LAYOUT1          =HTMLLayout
log4r.layout.html_layout1.conversionPattern=%5N: %r [%c] %-5p - %m%n



# ====================== define and configure filters
; ---------------------- filter named "WEEKDAYS"
log4r.filter.WeekDays              =WeekdayRangeFilter
log4r.filter.weekdays.weekdayMin   =Monday
log4r.filter.weekdays.weekdayMax   =Friday
log4r.filter.weekdays.acceptOnMatch=true

; ---------------------- filter named "WEEKEND"
log4r.filter.WeekEnd               =WeekdayRangeFilter
log4r.filter.weekend.weekdayMin    =Monday
log4r.filter.weekend.weekdayMax    =Friday
log4r.filter.weekend.acceptOnMatch =false

; ---------------------- filter named "WORKINGHOURS"
log4r.filter.WorkingHours          =TimeRangeFilter
log4r.filter.workinghours.timeMin  =08:00
log4r.filter.workinghours.timeMax  =18:00
log4r.filter.workinghours.acceptOnMatch=true
```

8 Appendix B. Example "simplelog4r.properties"

The log4r framework is distributed with an example "simplelog4r.properties" file which originally looked like this appendix.

```
; 2007-01-22, ---rgf, a "simplelog.properties" file, gets processed by the "SimpleLog" class and
;             entries are strings in the form: key=value
;             empty lines or lines starting with ';' or '#' are ignored


# ====================================================================================
; turn on 'log4r' framework debugging:
# log4r.config.LogLog.debug=1
;
; turn even logging of framework warnings and errors off:
# log4r.config.LogLog.quietMode=1
;
# ====================================================================================


# ====================================================================================
# ====================== define defaults for 'SimpleLog' loggers

; 'showDateTime',  values: 1 | [.]true or 0 [.]false
log4r.config.simplelog.showDateTime = 0

; 'showLogName',   values: 1 | [.]true or 0 [.]false
log4r.config.simplelog.showLoggerName = 0

; 'showShortName', values: 1 | [.]true or 0 [.]false
log4r.config.simplelog.showShortName = 0

; 'defaultLog', possible values: ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF
log4r.config.simplelog.defaultLogLevel = WARN


# ====================================================================================
# ====================== define 'SimpleLog' loggers

; 'logLevel' defaults to "INFO", values: ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF
;
; 'logLevel': log messages starting at the following level will be processed:
-- name of the following logger: "a_simple_logger_1"
log4r.logger.a_simple_logger_1               =

-- name of the following logger: "a_simple_logger_2"
log4r.logger.a_simple_logger_2               = warn
log4r.logger.a_simple_logger_2.showDateTime   = true
log4r.logger.a_simple_logger_2.showLoggerName = false
log4r.logger.a_simple_logger_2.showShortName  = true

-- name of the following logger: "a.simple.logger.3"
log4r.logger.a.simple.logger.3               = error
log4r.logger.a.simple.logger.3.showDateTime   = true
log4r.logger.a.simple.logger.3.showLoggerName = true
log4r.logger.a.simple.logger.3.showShortName  = true
```

9 Appendix C. The "rgf.sockets" Class Library

In the context of creating the log4r framework, there was a need for using TCP/IP sockets, in order to be able to create the `TelnetAppender`. Due to licensing issues the author decided to implement an ooRexx class for a TCP/IP socket and one for a TCP/IP socket servicing incoming connections (a "socket server").

The program "`rgf.sockets.cls`" contains both class definitions and is available in open source and for free. It wraps the external socket function library that comes with ooRexx, therefore the commented source code can be used together with the documentation of the "`RxSock`" package (cf. the PDF file named "`rxsock.pdf`" of the ooRexx distribution) and should fully document the APIs that are available to those who wish to employ the "`rgf.sockets`" class library.

To ease the mastering of these two classes figure 9.1 depicts the class hierarchy and the defined methods (class methods are shown in bold), such that the reader gains an immediate overview and is able to navigate to those methods in the source code which need to be studied.

This class library employs logging itself, and makes the logger object available via

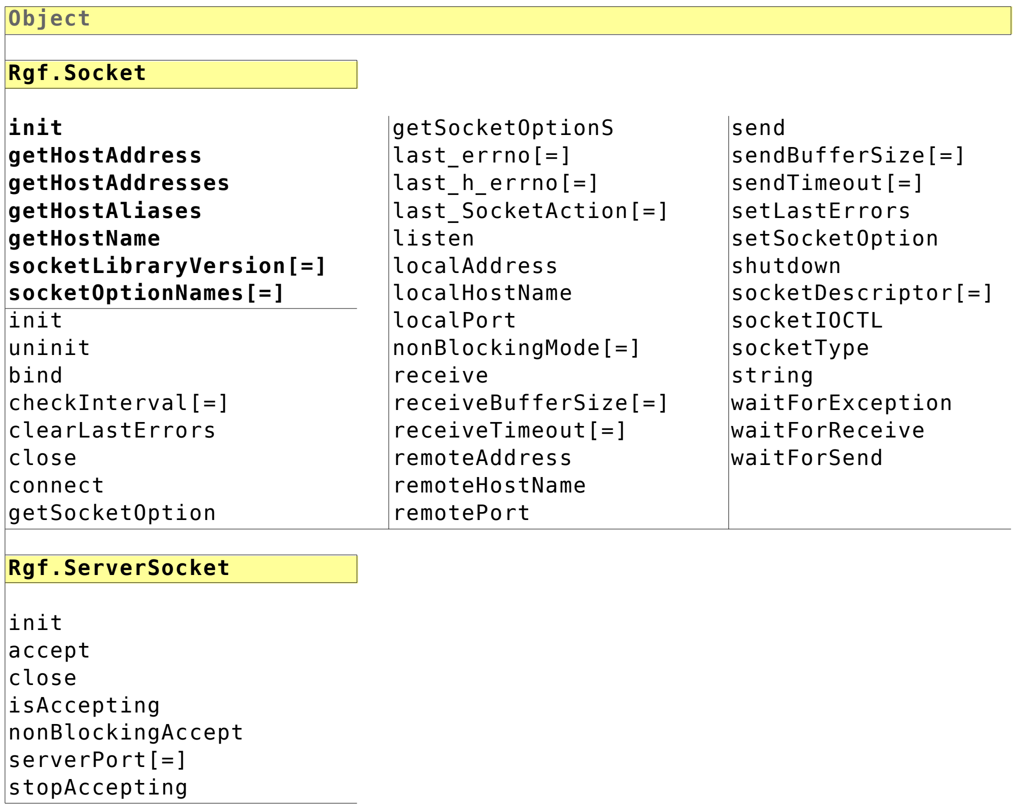


Figure 9.1: The "rgf.sockets" Class Hierarchy with the Defined Methods.

the environment symbol (note the leading dot!) "`.rgf.sockets.logger`" and if the `log4r` framework is available, this logger will be retrievable via the `.LogManager` class by the name "`rgf.sockets`", e.g. with the following statement:

```
logger=.LogManager~getLogger("rgf.sockets")
```

However, there is an "emergency logger implementation" built-in, such that the `log4r` framework needs not to be available for this class library to work!

Because of this, every Rexx program that uses this class library becomes able to use the class library's built-in log messages for its own debugging purposes!

Figure 9.2 depicts a simple Rexx program that tries to connect to the well defined http port of a non-existing host (wrongly spelled).

```
call rgf.sockets.cls       /* get the socket classes  */

s=.rgf.socket~new          /* get a socket            */
hostId="www.RexxLA.orgHHH" /* set hostname            */
port  =80                  /* set port number         */

res=s~connect(hostId, port)
say "res="res "(0=success, -1=error)"
```

Figure 9.2: Program '`t4socket.rex`" Using the "rgf.sockets.cls" Class Library.

The output of running the program in figure 9.2 is shown in figure 9.3.

```
   547 *-*     RAISE SYNTAX 93.900 array("'"address~string"' cannot be resolved!") -- not a positive, 
whole number
     7 *-* res=s~connect(hostId, port)
Error 93 running F:\work\log4ooRexx\rgf.sockets.cls line 547:  Incorrect call to method
Error 93.900:  'www.RexxLA.orgHHH' cannot be resolved!
```

Figure 9.3: Console Output of Running '`rexx t4socket.rex`".

The slightly changed program is depicted in figure 9.4. It activates the signal handling on any condition such that the program is not forcefully stopped in case of an error.

```
call rgf.sockets.cls       /* get the socket classes  */

s=.rgf.socket~new          /* get a socket            */
hostId="www.RexxLA.orgHHH" /* set hostname            */
port  =80                  /* set port number         */
signal on any
res=s~connect(hostId, port)
say "res="res "(0=success, -1=error)"
exit

any:
  say -2              /* indicate problem in the API   */
```

Figure 9.4: Program '`t5socket.rex`": Activates Signal Handling for Stopping Quietly..

The output of running the program in figure 9.4 is shown in figure 9.5.

```
-2
```

Figure 9.5: Console Output of Running '`rexx t5socket.rex`'.

The program in figure 9.6 demonstrates how easy it is to access the class library's logger via its environment symbol ("`.rgf.sockets.logger`").

By default the "`rgf.sockets`" class library creates a "no-operation" type of an "`EmergencyLogger`" in the case that the `log4r` framework is not loaded yet. If processing of log messages is desired, one may define the environment symbol "`.rgf.sockets.processLogs`" with a value of `.true`, which will cause the creation of an "`EmergencyLogger`" that processes the socket class library log messages.

After retrieving the logger its log level will be set to "`DEBUG`" to get to see all debug messages from this point in time on.[9]

```
   /* let the "rgf.sockets" class library know, that we want
      it to process log messages, even if 'log4r' is not available */
.local~rgf.sockets.processLogs=.true
call rgf.sockets.cls       /* get the socket classes  */
.rgf.sockets.logger~logLevel="debug"   /* activate processing for log messages >= "DEBUG" */

s=.rgf.socket~new          /* get a stream socket     */
hostId="www.RexxLA.orgHHH" /* set hostname            */
port  =80                  /* set port number         */
signal on any
res=s~connect(hostId, port)   /* connect the socket   */
say "res="res "(0=success, -1=error)"
exit

any:
  .rgf.sockets.logger~fatal("oops!", condition("o"))
  say -2            /* indicate problem in the API   */
```

Figure 9.6: Program '`t6socket.rex`': Activate Processing of '`DEBUG`' Log Messages.

The output of running the program in figure 9.6 is shown in figure 9.7.

```
Debug - 2007-04-25 13:41:59.589000: (a rgf.Socket: 4012F31F) - Socket.init | type=[], socketDescriptor=[]
Debug - 2007-04-25 13:41:59.589000: (a rgf.Socket: 4012F31F) - Socket.setLastErrors | last_ERRNO=[],
last_H_ERRNO=[], last_SocketAction=[]
Debug - 2007-04-25 13:41:59.589000: (a rgf.Socket: 4012F31F) - Socket.init | socketDescriptor=[196]
Debug - 2007-04-25 13:41:59.589000: (a rgf.Socket: 4012F31F) - Socket.connect | received
address=[www.RexxLA.orgHHH], port=[80]
Debug - 2007-04-25 13:41:59.589000: (a String: 9D454C46) - getHostInfos() | host=[www.RexxLA.orgHHH],
switch=[2]
Fatal - 2007-04-25 13:41:59.599000: a Directory - oops!
-2
Debug - 2007-04-25 13:41:59.599000: (a rgf.Socket: 4012F31F) - Socket.uninit | socketDescriptor=[196]
```

Figure 9.7: Console Output of Running '`rexx t6socket.rex`'.

Figure 9.8 just inserts the call to the `log4r` framework as the very first statement, before calling the "`rgf.sockets`" class library. As a result that class library now uses

[9] To later turn off the processing of log messages altogher one would merely need to state: `.rgf.sockets.logger~logLevel="OFF"`

a logger from the log4r framework configured according to the "log4r.properties" (or if not found: "simplelog4r.properties") file instead of its own one (dubbed "EmergencyLogger"). If the "log4r.properties" file is configured as in Appendix A above (cf. chapter 7, p. 40, logger configuration for the logger named "rgf.sockets"), then the log messages will also be sent to a HTML appender.

```
call load_log4r   /* load the 'log4r' framework first */

call rgf.sockets.cls       /* get the socket classes  */
.rgf.sockets.logger~logLevel="debug"    /* <---        */

s=.rgf.socket~new          /* get a stream socket     */
hostId="www.RexxLA.orgHHH" /* set hostname            */
port  =80                  /* set port number         */
signal on any
res=s~connect(hostId, port)   /* connect the socket   */
say "res="res "(0=success, -1=error)"
exit

any:
  .rgf.sockets.logger~fatal("oops!", condition("o"))
  say -2              /* indicate problem in the API   */
```

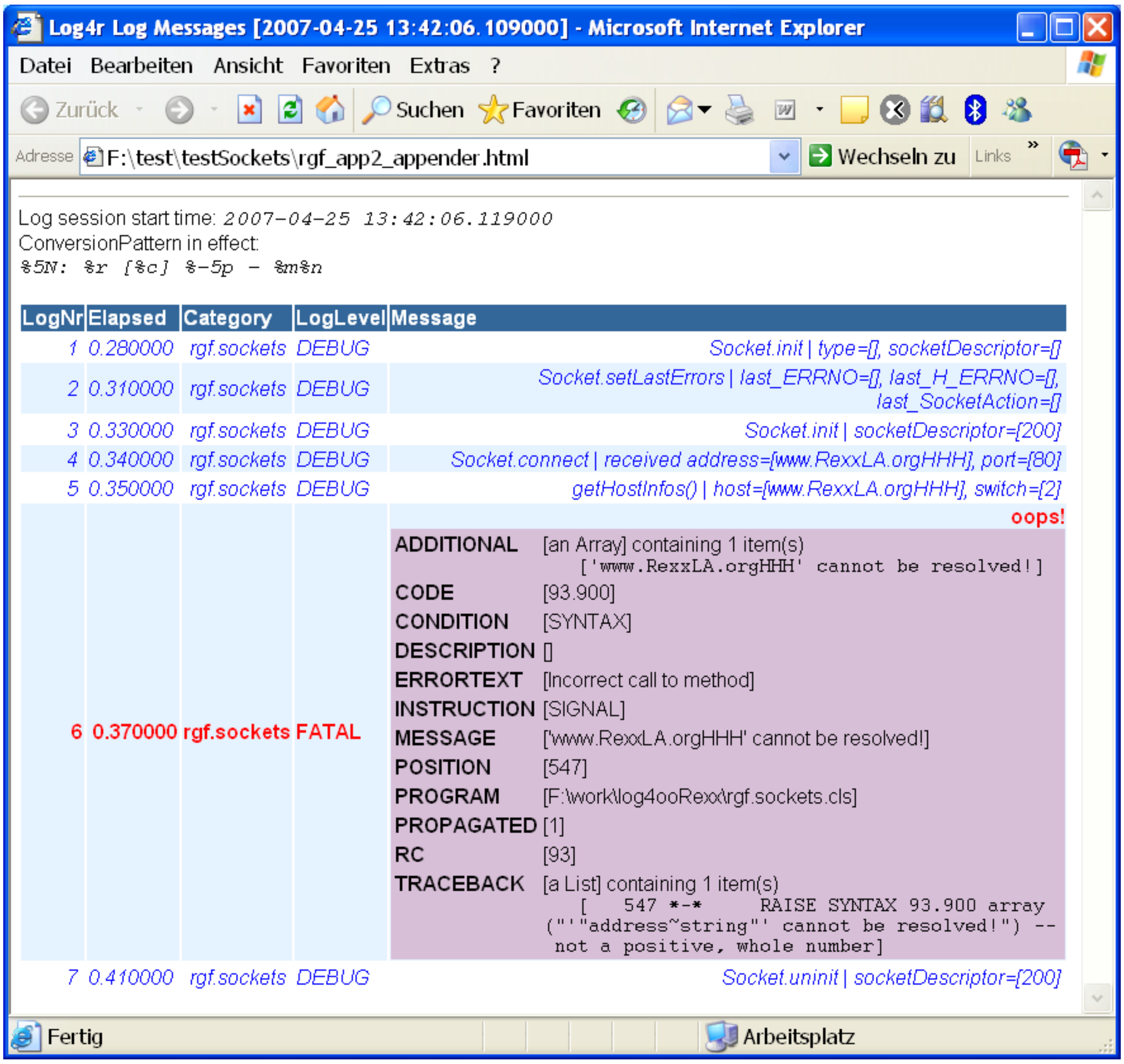
Figure 9.8: Program 't7socket.rex": Use the log4r Framework for Logging.

The output of running the program in figure 9.8 is shown in figure 9.9 and in figure 9.10.

```
    1: 0.280000 [rgf.sockets] DEBUG - Socket.init | type=[], socketDescriptor=[]
    2: 0.310000 [rgf.sockets] DEBUG - Socket.setLastErrors | last_ERRNO=[], last_H_ERRNO=[],
last_SocketAction=[]
    3: 0.330000 [rgf.sockets] DEBUG - Socket.init | socketDescriptor=[200]
    4: 0.340000 [rgf.sockets] DEBUG - Socket.connect | received address=[www.RexxLA.orgHHH], port=[80]
    5: 0.350000 [rgf.sockets] DEBUG - getHostInfos() | host=[www.RexxLA.orgHHH], switch=[2]
    6: 0.370000 [rgf.sockets] FATAL - oops!
                        ADDITIONAL..[an Array] containing 1 item(s)
                                         --> ['www.RexxLA.orgHHH' cannot be resolved!]
                        CODE........[93.900]
                        CONDITION...[SYNTAX]
                        DESCRIPTION.[]
                        ERRORTEXT...[Incorrect call to method]
                        INSTRUCTION.[SIGNAL]
                        MESSAGE.....['www.RexxLA.orgHHH' cannot be resolved!]
                        POSITION....[547]
                        PROGRAM.....[F:\work\log4ooRexx\rgf.sockets.cls]
                        PROPAGATED..[1]
                        RC..........[93]
                        TRACEBACK...[a List] containing 1 item(s)
                                         --> [   547 *-*     RAISE SYNTAX 93.900
array("'"address~string"' cannot be resolved!") -- not a positive, whole number]

-2
    7: 0.410000 [rgf.sockets] DEBUG - Socket.uninit | socketDescriptor=[200]
```

Figure 9.9: Console Output of Running 'rexx t7socket.rex".


Log4r Log Messages [2007-04-25 13:42:06.109000] - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse
F:\test\testSockets\rgf_app2_appender.html
Wechseln zu
Links
Fertig
Arbeitsplatz

Log session start time: `2007-04-25 13:42:06.119000`
ConversionPattern in effect:
`%5N: %r [%c] %-5p - %m%n`

LogNr	Elapsed	Category	LogLevel	Message
1	0.280000	rgf.sockets	DEBUG	Socket.init | type=[], socketDescriptor=[]
2	0.310000	rgf.sockets	DEBUG	Socket.setLastErrors | last_ERRNO=[], last_H_ERRNO=[], last_SocketAction=[]
3	0.330000	rgf.sockets	DEBUG	Socket.init | socketDescriptor=[200]
4	0.340000	rgf.sockets	DEBUG	Socket.connect | received address=[www.RexxLA.orgHHH], port=[80]
5	0.350000	rgf.sockets	DEBUG	getHostInfos() | host=[www.RexxLA.orgHHH], switch=[2]
6	0.370000	rgf.sockets	FATAL	oops! **ADDITIONAL** [an Array] containing 1 item(s) `['www.RexxLA.orgHHH' cannot be resolved!]` **CODE** [93.900] **CONDITION** [SYNTAX] **DESCRIPTION** [] **ERRORTEXT** [Incorrect call to method] **INSTRUCTION** [SIGNAL] **MESSAGE** ['www.RexxLA.orgHHH' cannot be resolved!] **POSITION** [547] **PROGRAM** [F:\work\log4ooRexx\rgf.sockets.cls] **PROPAGATED** [1] **RC** [93] **TRACEBACK** [a List] containing 1 item(s) `[ 547 *-* RAISE SYNTAX 93.900 array ("'"address~string"' cannot be resolved!") -- not a positive, whole number]`
7	0.410000	rgf.sockets	DEBUG	Socket.uninit | socketDescriptor=[200]

Figure 9.10: The HTML Appender Output of Running '`rexx t7socket.rex`'.