

Templates

The most important part of the preferences are the templates. They affect which logger should be used.

Templates are defined in the "Log4E" preferences root page. Several predefined templates are available (Log4j, Jakarta Commons Logging, JDK 1.4 Logging and ATG Logging).

Options:

- Show

Shows ready-made template definitions. These views are read only. Modifications are not saved!

- Edit [Pro version only]

Opens a template definition for selfdefined templates which can be edited. The modification will be saved immediately when pressing "OK".

- Duplicate [Pro version only]

One can duplicate a template to adapt it to his own logger.

- Rename [Pro version only]

Rename a selfdefined template.

- Remove [Pro version only]

Only selfdefined templates are allowed to remove. To confirm the deletion one must press "OK" or "Apply".
Template Definitions

After pressing "Show" or "Edit" a popup window with the definitions is shown.

Sample Template definition:

```
{logger}.error("${enclosing_method}${delimiter}${message}${delimiter}
${message_user}${delimiter}${variables}${delimiter}${return_value}", ${exception})
```

Variables

- \${delimiter}

The delimiter is defined in Log4E > Format preference page. It is very important when using the "Modification" task.

- \${delimiter_msg}

The delimiter_msg is defined in Log4E > Format preference page. It is used when invoking "Insert Log Statements At This Position..." to separate the informations within a message (the message itself and the variables of the enclosing method). The default is " : ". It should be different to \${delimiter}

Example:

```
logger.debug("myMethod() - my message : int i = " + i);
```

- \${enclosing_method}

The given enclosing method WITH arguments. It is expected that the variable is embedded in a String context.

e.g. myMethod(String str = " + str + ")

- \${enclosing_method_arguments}

The arguments of the given method (\${enclosing_method_only}) . It is expected that the variable is embedded in a String context.

e.g. String str = " + str

- \${enclosing_method_only}

The given enclosing method WITHOUT arguments. e.g. myMethod

- \${enclosing_package}

The given package.

- \${enclosing_type}

The given enclosing class.

- \${exception}

The given exception in a catch clause.

- \${logger}

The "logger name" is defined in the Declaration Tab of the Log4E > Imports and Declaration preference page.

- \${message}

The message is defined in Log4E > Positions preference page.

- \${message_user}

It is a user generated message. It is used by the "Substitute" und "Log this position..." tasks.

- \${return_value}

The given return value of the method. It is replaced only if Log4E > Positions > Method Exit > Move invocation ... is

checked.

- `${variables}`

It decouples `${message}` from local vars (generated in "Log this position...").

Declaration Tab

Options:

- Imports

defining logger imports. example: "org.apache.log4j.Logger"

- Initializer

Declaration and Initializing the Logger for a class.

Example:

```
/**
 * Logger for this class
 */
private static final Logger ${logger} =
    Logger.getLogger(${enclosing_type}.class)
If this field is left blank, the logger won't be declared.
```

As a result the logger imports and declaration is automatically inserted like in that example:

```
import org.apache.log4j.Logger;
public class MyClass { /**
 * Logger for this class
 */
private static final Logger logger = Logger.getLogger(MyClass.class);
```

Note:

In this example the Logger is declared as static. Also the default templates declare the logger as static. Please be aware that approach might not fit for you, especially when you work in a shared environment like an application in an servlet engine (e.g. Tomcat) or application server (e.g. Weblogic).

More information at [SLF4J FAQ](#) and the [commons-logging WIKI](#). [Level Statements Tab](#)

You can specify level statements for any of these levels:

FINEST, FINER, TRACE, DEBUG, INFO, WARN, ERROR, FATAL.

The denotation of the levels is a mixture between Commons Logging and JDK 1.4 logging denotation. FINEST is the lowest level and is only used in JDK 1.4 logging. FATAL is the highest level and is equivalent to SEVERE in JDK 1.4 logging.

Note that these level and methods are not available for all loggers (e.g. Log4j doesn't support TRACE, but the Apache Commons Logger does).

Options:

- `trace()` Statement

the method which should be invoked if level is TRACE.

Example:

```
${logger}.trace("${enclosing_method}${delimiter}
 ${message}${delimiter}${message_user}${delimiter}${variables}${delimiter}${return_value}")
```

As a result the logger statements are automatically inserted like in that example:

```
public String myMethod(String arg1) {
    logger.debug("myMethod() - start");

    //Your code....

    logger.debug("myMethod() - end - return value = myString");
    return "myString";
}
```

[Is<Level>Enabled Statements Tab](#)

Define the methods which check the level before executing the log statement here.
 These statements are only used if specified in the [Log4E > Statements preference page](#).

Options:

- `isTraceEnabled()` Statement
 the method which checks the if logger level is TRACE.

Example:

```
${logger}.isTraceEnabled() or ${logger}.isLoggable(Level.FINER)
```

As a result the logger statements are automatically inserted like in that example:

```
public String myMethod(String arg1) {
  if (logger.isDebugEnabled()) {
    logger.debug(...);
  }
}
```

```
//Your code....
}
```

Position Statements Tab

"Position Statements" is an alternative to the level statements described above. They are introduced in JDK 1.4 logging and are special statements for particular method entries:

Options:

- Start Statement

E.g `${logger}.entering("${enclosing_type}", "${enclosing_method}", "${message}${delimiter}${message_user}")` in JDK 1.4 logging. It will be inserted at start position if [Log4E > Positions > Method Start > Use position log statements ...](#) is checked.

- End Statement

E.g `${logger}.exiting("${enclosing_type}", "${enclosing_method}", "${message}${delimiter}${message_user}${delimiter}${return_value}")` in JDK 1.4 logging. It will be inserted at end position if [Log4E > Positions > Method Exit > Use position log statements ...](#) is checked.

- Throwing Statement

E.g `${logger}.throwing("${enclosing_type}", "${enclosing_method}", ${exception})` in JDK 1.4 logging. It will be inserted at catch position if [Log4E > Positions > Catch Block > Use position log statements ...](#) is checked.

As a result the logger statements are automatically inserted like in that example:

```
public String myMethod(String arg1) {
  logger.entering("MyClass", "myMethod()", "start");
```

```
//Your code....
logger.entering("MyClass",
  "myMethod()", "end - return value = myString");
return "myString";
}
```

Is<Position>Enabled Statements Tab

These statements are equivalent to the [Is<Level>Enabled Statements](#) described above.

Options:

- `isStartEnabled` Statement

E.g. `${logger}.isLoggable(Level.FINER)`

This statement is only used if specified in the [Log4E > Statements preference page](#).

- `isEndEnabled` Statement

E.g. `${logger}.isLoggable(Level.FINER)`

This statement is only used if specified in the [Log4E > Statements preference page](#).

- `isThrowingEnabled` Statement

E.g. `${logger}.isLoggable(Level.FINER)`

This statement is only used if specified in the [Log4E > Statements preference page](#).

As a result the logger statements are automatically inserted like in that example:

```
public String myMethod(String arg1) {  
    if (logger.isLoggable(Level.FINER)) {  
        logger.entering(...);  
    }  
  
    //Your code....  
}
```