

Positions

Log4E inserts logger statements at certain method entries. You can specify the behaviour for each entry differently. The entries are separated in different tabs.

Tab Method Start

It defines the very beginning of the method. The logger statement is inserted there. If method is a constructor and does a constructor invocation (i.e. this(), super()) the logger statement is inserted right behind the invocation.

Options:

- Skip logging for this position

If checked logger statements are not inserted at the beginning of method.

- Skip in Constructor

If Log4E is invoked for the whole class, it leaves out constructor methods

- Skip methods with empty body [Pro version only]

If checked logger statements are not inserted in empty methods.

- Skip getter [Pro version only]

If checked logger statements are not inserted in getter methods.

The plugin identifies any method that looks like that:

```
public String anyMethodName() {
    return "mystring";
}
```

The method in the example doesn't even have a prefix "get" but has all qualities of a getter method.

- Skip setter [Pro version only]

If checked logger statements are not inserted in setter methods.

The plugin identifies any method that looks like that:

```
public void anyMethodName(String mystring) {
    this.mystring = mystring;
}
```

The method in the example doesn't even have a prefix "set" but has all qualities of a setter method.

- Use position log statements instead of level statements

Use the alternative position statements. They are defined in [Log4E > Templates > Position Statements](#)

- Choose level

Defines the level of logger. The recommended level for this position is DEBUG.

- Insert info about current method

if disabled, current method infos are not provided

- Log with parameter types of method

Argument types of method

Example:logger.debug("myMethod(String) - start");

- Log with parameter names of method

Argument names of method

Example:logger.debug("myMethod(arg1) - start");

- Log with parameter values of method

Argument values of method

Example:logger.debug("myMethod(" + arg1 + ") - start");

- Logger message

Your message after method name

Example:logger.debug("myMethod(String) - start");

As a result the logger statements are automatically inserted like in that example: public String myMethod(String arg1) {

```
if (logger.isDebugEnabled()) {
    logger.debug("myMethod(String arg1 = " + arg1 + ") - start");
}
```

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

```
}
```

Tab Method Exit

It defines the exit point of a method. That could be a return statement or the end of the method. Therefore a logger statement is inserted before every return statement and at the end of method.

The settings are the same as at start position except these:

Options:

- Move invocation in return statement in front of logger
 - Include return value
- Inserts the return value in logger statement

Example:

Before:

```
public String myMethod(String arg1) {
//Your code....
return toString();
}
```

After:

```
public String myMethod(String arg1) {
//Your code....
String returnString = toString();
logger.debug("myMethod() - end - return value = " + returnString);
return returnString;
}
```

Tab Catch Block

It defines the position in a try - catch block. Normally a logger with ERROR level is inserted here providing the given exception.

The settings are the same as at start position. Some additional options are available:

- Skip empty catch block [Pro version only]

If checked, logger statements are not inserted in empty catch blocks. If unchecked, the options "Level for empty catch block" and "Message in empty catch block (\${message})" will be used instead.

- Level for empty catch block [Pro version only]

Choose a level that should be used in empty catch blocks. This can be different from "Choose level". Default: "Log4j Warn/JDK14 Warning"

- Message in empty catch block (\${message}) [Pro version only]

Choose a message that should be used in empty catch blocks. This can be different from "Logger message". Default: "exception ignored"

- Log all variables available [Pro version only]

If checked all variables available in the scope of the current catch block will be logged.

Tab Other

It defines all other positions than "Method Start", "Method Exit" or "Catch Block".

This occurs when System out's are replaced at other positions or when log statements are modified at other positions or if you use the "Insert Log Statement At This Position..." task.

The settings are reduced to the needed information.