message Plugin

Table of contents

1 Message plugin 3
2 Parameters 3
3 Extension point message.start 3
3.1 Attribute file 4
3.2 Attribute append 4
4 Extension point message.end 4
4.1 Attribute file 4
4.2 Attribute append 5
5 Listener message.listener 5
5.1 Attribute file 5
5.2 Attribute append 5
6 Element br 6
7 Element print 6
7.1 Attribute symbol 6
7.2 Attribute library 6
7.3 Attribute import 7
8 Element arg 7
8.1 Attribute index 7
8.2 Attribute symbol 7
8.3 Attribute library 7
8.4 Attribute import 8
8.5 Attribute format 8
9 Export print.time.now 8
10 Variable event.id 8

11 Variable event.num-args..... 8

1. Message plugin

The message plugin allows messages to be sent to files at various points during the execution of the application.

For example, the following code in a plugin will print `Hello, World!` after the application starts and `Goodbye, World!` before it finishes. It also prints information every time a `com.acme.some-event` fires. All of these printings are made to file `log.txt`.

```
<?xml version="1.0"?>
<?plugin version="0.1.0"?>
<plugin id="test.message.plugin">
        <extension point="message.start" file="log.txt">Hello,
World!<br/></extension>
        <extension point="message.stop" file="log.txt">Goodbye,
World!<br/></extension>
        <extension point="com.acme.some-event" listener="message.listener"
file="log.txt">
The ${event.id} event has just fired at <print import="print.time.now/>.
It had ${event.num-args} arguments.
Argument 0, which must be a string for this to work, was "<arg
import="print.string"/>".
        </extension>
</plugin>
```

2. Parameters

Parameter	Value	Notes
Id	message	
Version processing instruction	<?plugin version="0.1.0"?>	
Version	0.1.0	
Location	message-0.1.0	
Lazy	true	
Author	Hugh Leather	

3. Extension point message.start

```
<extension point="message.start"
    file?    = file-name default "/dev/stdout"
    append?  = boolean default "true">
```

```
    <!-- Content: ( PCDATA | print | br )* -->
</extension>
<!-- Contained by: plugin -->
```

By extending the point, `message.start`, plugins can have text written to a file after the application has started.

Text will be variable expanded.

3.1. Attribute file

This attribute determines which file the text will be output to. By default it will print text to the standard output.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

3.2. Attribute append

This attribute determines if the file will be appended to or created afresh. By default the file will be appended to.

> **Note:**
> Creating the file afresh can cause missing data if multiple plugins write to the same file since the order of extensions executing is not defined.

The attribute is variable expanded in the context of the extending plugin.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

4. Extension point message.end

```
<extension point="message.end"
    file?    = file-name default "/dev/stdout"
    append?  = boolean default "true">
    <!-- Content: ( PCDATA | print | br )* -->
</extension>
<!-- Contained by: plugin -->
```

By extending the point, `message.end`, plugins can have text written to a file before the application stops.

Text will be variable expanded.

4.1. Attribute file

This attribute determines which file the text will be output to. By default it will print text to

the standard output.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

4.2. Attribute append

This attribute determines if the file will be appended to or created afresh. By default the file will be appended to.

> **Note:**
> Creating the file afresh can cause missing data if multiple plugins write to the same file since the order of extensions executing is not defined.

The attribute is variable expanded in the context of the extending plugin.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

5. Listener message.listener

```
<listener import="message.listener"
    file?    = file-name default "/dev/stdout"
    append?  = boolean default "true">
    <!-- Content: ( PCDATA | print | arg | br )* -->
</extension>
<!-- Contained by: extension for event -->
```

This exports an event listener that can be used to print (basic) information about events to files.

Text will be variable expanded. Additional variables are provided: `event.id` and `event.num-args`.

5.1. Attribute file

This attribute determines which file the text will be output to. By default it will print text to the standard output.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

5.2. Attribute append

This attribute determines if the file will be appended to or created afresh. By default the file will be appended to.

Note:
Creating the file afresh can cause missing data if multiple plugins write to the same file since the order of extensions executing is not defined.

The attribute is variable expanded in the context of the extending plugin.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

6. Element br

```
<br>
    <!-- Content: EMPTY -->
</br>
<!-- Contained by: message.start, message.stop, message.listener -->
```

Prints a new line.

7. Element print

```
<print
    symbol?  = symbol name
    library? = library id
    import?  = extension point id>
    <!-- Content: EMPTY -->
</print>
<!-- Contained by: message.start, message.stop, message.listener -->
```

A print element calls a function to print text to the file. The function must have signature:

```
void (*)( FILE* file )
```

7.1. Attribute symbol

The symbol attribute gives the name of a symbol for the function in one of the extending plugin's libraries. The particular library can be specified with the `library` attribute.

This attribute may not be used if `import` is used.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

7.2. Attribute library

The library attribute gives the name of the library in which a symbol for the function is found.

The attribute must not be used unless attribute, `symbol`, is also given.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

7.3. Attribute import

The import attribute gives the name of a symbol exported by another plugin.

This attribute may not be used if `symbol` is used.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

8. Element arg

```
<arg
    index    = positive integer
    symbol?  = symbol name
    library? = library id
    import?  = extension point id
    format?  = string>
    <!-- Content: EMPTY -->
</arg>
<!-- Contained by: message.start, message.stop, message.listener -->
```

This element prints the value of an argument to the file. The printer function is given by an import or by a symbol in a library of this plugin.

The printer function must have the following prototype:

```
void (*)( FILE* file, void* value )
```

8.1. Attribute index

The index of the argument to print

This attribute is variable expanded. Variables are resolved in the context of the plugin.

8.2. Attribute symbol

The symbol attribute gives the name of a symbol for the function in one of the extending plugin's libraries. The particular library can be specified with the `library` attribute.

This attribute may not be used if `import` or `format` is used.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

8.3. Attribute library

The library attribute gives the name of the library in which a symbol for the function is

found.

The attribute must not be used unless attribute, `symbol`, is also given.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

8.4. Attribute import

The import attribute gives the name of a symbol exported by another plugin.

This attribute may not be used if `symbol` or `format` is used.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

8.5. Attribute format

This attribute allows something similar to printf to be acheived. The format string is that part of a printf format string after a %. No other characters are permitted.

For example: `<arg index="1" format="4.2f"/>` will assume that the 1st argument is a float and prints it as you might expect from `printf( "%4.2f", arg )`.

This attribute may not be used if `import` or `symbol` is used.

This attribute is variable expanded. Variables are resolved in the context of the plugin.

9. Export print.time.now

This export is a function of signature: `void (*)( FILE* file )`

It will print the current time to the given file (which must be open)

10. Variable event.id

This variable expands to the id of the event that is fired.

11. Variable event.num-args

This variable expands to the number of arguments for the event.