

# **Chromattic**

***Chromattic Reference Technical  
Documentation***



**Alain Defrance**

**eXo Platform**

Copyright © 2010 eXo Platform SAS



## Table of Contents

- 1. Groovy implementation
  - 1.1. Groovy port motivation
  - 1.2. AST Transformation
  - 1.3. Instrumentation
  - 1.4. Instanciation
  - 1.5. Chromatic core delegation

## List of Examples

- 1.1. The GroovyInstrumentor class
- 1.2. The GroovyProxyType class
- 1.3. The ChromaticGroovyInvocation class



# Groovy implementation

## 1.1. Groovy port motivation

The Groovy integration with Chromatic serves two purposes

- Chromatic relies a lot on annotation processing available in the Java compiler at compile time, while the same feature is available in the Groovy compiler, the transformation performed are not the same.
- Follow Groovy best principles and idioms to make the integration the most natural possible according to what Groovy developers are used to.

## 1.2. AST Transformation

Thanks to Abstract Syntax Tree Transformation (AST), the program is transformed to adapt the metamodel at compile time without requiring developer intervention. In fact, through AST transformation a simple Groovy class containing Chromatic annotated properties is adapted to Chromatic model. The following operations are made at compile time to have a well formed Chromatic model:

- Create or modify getter and setter methods to comply to Chromatic
- Move field annotations to getter and/or setter annotations.
- Add a field `chromaticInvoker` of type `org.chromatic.spi.instrument.MethodHandler` (The same as the Java version).
- Create a getter for the field `chromaticInvoker` (used by the instrumentor to delegate the call to chromatic engine).
- Create or modify the default constructor to provide a protected accessibility.
- Create an initializer constructor for the field `chromaticInvoker`.
- Do implement the `GroovyInterceptable` interface by the class.
- Override the `invokeMethod(String methodName, Object parameters)` method



(Delegate at the MOP level to the `chromatticInvoker` instance).

- Override the `getProperty(String propertyName)` and `setProperty(String propertyName, Object propertyValue)`

A simple input source file :

```
package org.chromattic.docs.technical.groovy

import org.chromattic.api.annotations.Name
import org.chromattic.api.annotations.Property
import org.chromattic.api.annotations.PrimaryType

/**
 * @author <a href="mailto:alain.defrance@exoplatform.com">Alain Defrance</a>
 * @version $Revision$
 */
@PrimaryType(name = "gs:page")
class Page {
    /**
     * The page name.
     */
    @Name def String name ❶

    /**
     * The page title.
     */
    @Property(name = "title") def String title ❷

    /**
     * The page content.
     */
    @Property(name = "content") def String content ❸
}
```

- ❶ The name property is mapped to the node name
- ❷ The title property is mapped to the title node property
- ❸ The content property is mapped to the content node property

The output source code after the AST Transformation :

```
package org.chromattic.docs.technical.groovy

import org.chromattic.api.annotations.Name
import org.chromattic.api.annotations.Property
import org.chromattic.api.annotations.PrimaryType
import org.chromattic.spi.instrument.MethodHandler
import org.chromattic.groovy.ChromatticGroovyInvocation

/**
 * @author <a href="mailto:alain.defrance@exoplatform.com">Alain Defrance</a>
 * @version $Revision$
 */
```



```

@PrimaryType(name = "gs:page")
class CompiledPage implements GroovyInterceptable { ❶
    /**
     * The page name.
     */
    ❷
    def String name

    /**
     * The page title.
     */
    def String title

    /**
     * The page content.
     */
    def String content

    private MethodHandler chromatticInvoker_; ❸

    protected CompiledPage() {} ❹
    public CompiledPage(MethodHandler chromatticInvoker) { ❺
        this.chromatticInvoker_ = chromatticInvoker;
    }

    ❻
    @Name
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Property(name = "title")
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @Property(name = "content")
    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content
    }

    ❼
    public Object invokeMethod(String m, Object p) {
        ChromatticGroovyInvocation.invokeMethod(this, m, p, chromatticInvoker)
    }

    ❽
    public Object getProperty(String p) {

```



```

    return ChromaticGroovyInvocation.getProperty(this, p, chromaticInvoker)
}

public void setProperty(String p, Object v) {
    ChromaticGroovyInvocation.setProperty(this, p, v, chromaticInvoker)
}
}

```

- ❶ Implements GroovyInterceptable
- ❷ The annotations were moved to the getter
- ❸ The method handler was created
- ❹ Default constructor become protected
- ❺ Initializer constructor was created
- ❻ Getter and setter has been generated
- ❼ Create invokeMethod method for MOP interception
- ❽ Create getProperty & setProperty method for MOP interception

## 1.3. Instrumentation

The Groovy port of chromatic requires its own instrumentor : `org.chromatic.groovy.instrument.GroovyInstrumentor` which allows to get the method handler of a given proxy instance (in Groovy, the proxy is the model instance).

### Example 1.1. The GroovyInstrumentor class

```

/**
 * @author <a href="mailto:alain.defrance@exoplatform.com">Alain Defrance</a>
 * @version $Revision$
 */
public class GroovyInstrumentor implements Instrumentor {
    public <O> ProxyType<O> getProxyClass(Class<O> clazz) {
        return new GroovyProxyType(clazz);
    }

    public MethodHandler getInvoker(Object proxy) {
        try {
            return (MethodHandler) proxy.getClass().getMethod("getChromaticInvoke:
        }
        catch (NoSuchMethodException e) {
            throw new IllegalArgumentException(e.getMessage(), e);
        } catch (Exception e) {
            throw new AssertionError(e);
        }
    }
}

```



## 1.4. Instanciation

The instantiation was made by `org.chromattic.groovy.instrument.GroovyProxyType<O>` class. This factory use the initializer constructor to initialize the `chromatticInvoker` field with the Chromattic method handler (generated at compile time).

### Example 1.2. The GroovyProxyType class

```
/**
 * @author <a href="mailto:alain.defrance@exoplatform.com">Alain Defrance</a>
 * @version $Revision$
 */
public class GroovyProxyType<O> implements ProxyType<O> {

    /** . */
    private final Constructor<? extends O> ctor;

    public GroovyProxyType(Class<O> clazz) {
        try {
            ctor = clazz.getConstructor(MethodHandler.class);
        } catch (Exception e) {
            throw new AssertionError(e);
        }
    }

    public O createProxy(MethodHandler handler) {
        try {
            return ctor.newInstance(handler);
        }
        catch (Exception e) {
            throw new AssertionError(e);
        }
    }

    public Class<? extends O> getType() {
        return ctor.getDeclaringClass();
    }
}
```

## 1.5. Chromattic core delegation

Any method invocation is redirected to a single entry point : the `org.chromattic.groovy.ChromatticGroovyInvocation` that forwards the Meta Object Protocol (MOP) call to the `MethodHandler`.

### Example 1.3. The ChromatticGroovyInvocation class

```
/**
 * @author <a href="mailto:alain.defrance@exoplatform.com">Alain Defrance</a>
 * @version $Revision$
```



```

*/
public class ChromatticGroovyInvocation {
    public static Object getProperty(Object target, String m, MethodHandler handler) {
        return invokeMethod(target, GroovyUtils.getSetName(GroovyUtils.GetSet.GET, m), handler, null);
    }

    public static Object setProperty(Object target, String m, Object v, MethodHandler handler) {
        return invokeMethod(target, GroovyUtils.getSetName(GroovyUtils.GetSet.SET, m), handler, v);
    }

    public static Object invokeMethod(Object target, String m, Object p, MethodHandler handler, Method method) {
        try {
            method = target.getClass().getMethod(m, args2Class(p));
        } catch (NoSuchMethodException _) {
            try {
                method = foundMethod(target.getClass(), m, p);
            } catch (NoSuchMethodException __) {
                // If method cannot be found, the method is getter or setter and the field exists
                // We directly access to it.
                try {
                    Field field = target.getClass().getField(GroovyUtils.fieldName(m));
                    return field.get(target);
                } catch (Exception e) {
                    throw new AssertionError(e);
                }
            }
        }

        // method exist
        try {
            if (isChromatticAnnotated(method)) {
                return handler.invoke(target, method, (Object[]) p);
            } else {
                return method.invoke(target, (Object[]) p);
            }
        } catch (RuntimeException re) {
            throw re;
        } catch (Error e) {
            throw e;
        } catch (Throwable t) {
            throw new AssertionError(t);
        }
    }

    private static Class[] args2Class (Object objs) {
        List<Class> classes = new ArrayList<Class>();
        for (Object o : (Object[]) objs) {
            classes.add((o != null ? o.getClass() : Object.class));
        }
        return classes.toArray(new Class[]{});
    }

    private static Method foundMethod(Class<?> from, String m, Object args) throws NoSuchMethodException {
        for (Method method : from.getMethods()) {
            if (
                method.getName().equals(m)
                && method.getParameterTypes().length == ((Object[])args).length
                // TODO : maybe check parameter type
            ) return method;
        }
    }
}

```



```
    }  
    throw new NoSuchMethodException("Cannot find " + from.getName() + "." + m  
}  
  
public static boolean isChromaticAnnotated(Method method) {  
    for (Annotation annotation : method.getAnnotations()) {  
        if (annotation.toString().startsWith(GroovyUtils.ANNOTATIONS_PACKAGE, 1)  
    }  
}
```



```
    return false;  
  }  
}
```