Quick start guide

This guide will help you execute a custom rule analysis in Kiuwan Analyzer.

Contents:

If you need more information on programming a rule, please check the Getting Started with Rule Development page.

Prerequisites

For rule development you need the following prerequisites:

1. A suitable Kiuwan account for developing rules.
2. Kiuwan Local Analyzer (including **Kiuwan Rule Developer**)
   a. If you have already downloaded the analyzer, but you cannot find a development folder, please upgrade or reinstall the Kiuwan Local Analyzer.
3. Java 8 (64 bits) or above (JDK or JRE)
4. Your preferred code editor (Eclipse, iDEA, Vi, Emacs...)

Step 1: Preparing your IDE

The rules that Kiuwan executes are created using the Java language. A Kiuwan rule is composed by:

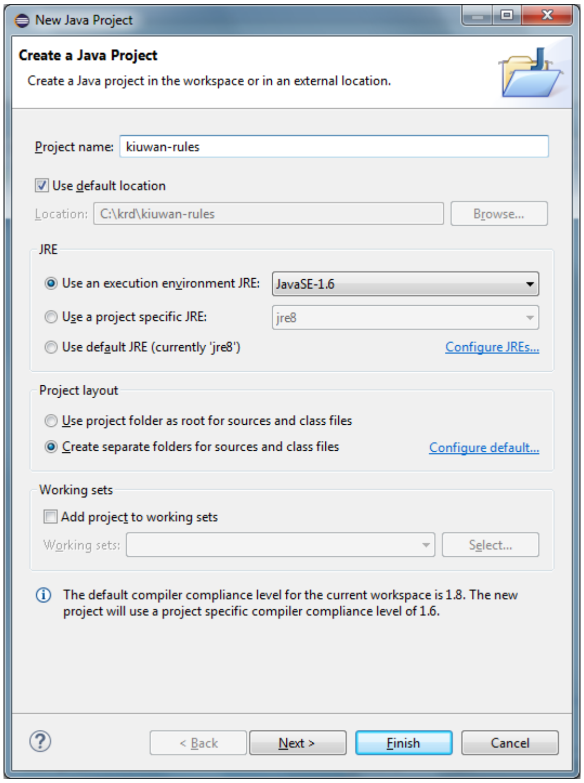
- A rule definition: an XML file where the rule metadata is kept.
- The rule source code, which contains the Java code that will be executed when running the rule.

In this section, we will create a Java project in Eclipse that will contain both parts of the rule. We will use for this guide the Eclipse Luna Standard, but you can use any Eclipse version or any other IDE or editor you like following an equivalent process:

1. Create a new Java project choosing File > New > Java Project...
   a. Choose a name for your project.
   b. Set the Java execution environment to **JavaSE-1.8.**
2. Click Next.
3. Configure the project libraries needed to create Kiuwan rules:
   a. Go to the **Libraries** tab.
   b. Click the **Add External JARs...** button.
   c. Go to the folder where you installed Kiuwan Local Analyzer (AGENT_HOME) and click on the **lib.engine** subfolder.
   d. Choose the **analyzer.jar** and the **spring.jar** files.
   e. Once the jar files are added to the libraries, click on the analyzer.jar drop-down icon so the jar configuration can be seen.
   f. Select the **Javadoc location** entry and click **Edit.**
   g. Click the **Javadoc in the archive** and then the **External file** radio buttons.
   h. Click **Browse..** and look for the AGENT_HOME/development/doc/javadoc/**analyzer-javadoc.zip** file
   i. Click **Open** and then **OK**.
   j. Click the **Finish** button in the New Java Project wizard to end the project configuration.
4. Once the project has been created, create a new folder called **definitions** under the new project. We will use this folder to store the rule definitions created by Kiuwan Rule Developer.

Step	Screenshot

1.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: kiuwan-rules

☑ Use default location

Location: C:\krd\kiuwan-rules Browse...

JRE

● Use an execution environment JRE: JavaSE-1.6

○ Use a project specific JRE: jre8

○ Use default JRE (currently 'jre8') Configure JREs...

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files Configure default...

Working sets

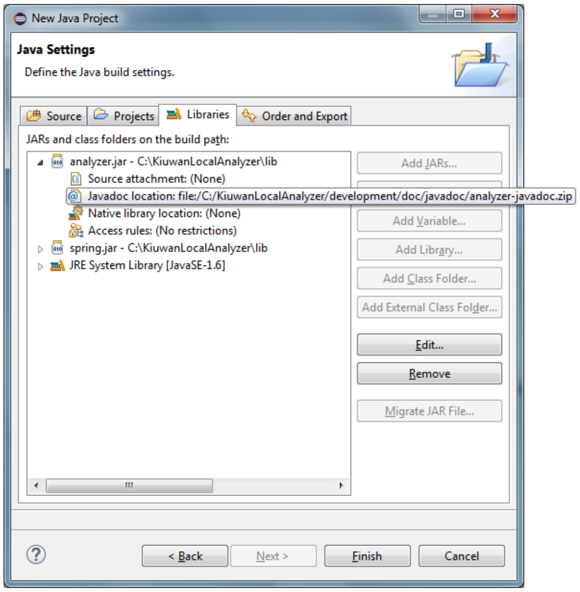
□ Add project to working sets

Working sets: Select...

ⓘ The default compiler compliance level for the current workspace is 1.8. The new project will use a project specific compiler compliance level of 1.6.

< Back Next > Finish Cancel

2.



New Java Project

Java Settings

Define the Java build settings.

Source | Projects | Libraries | Order and Export

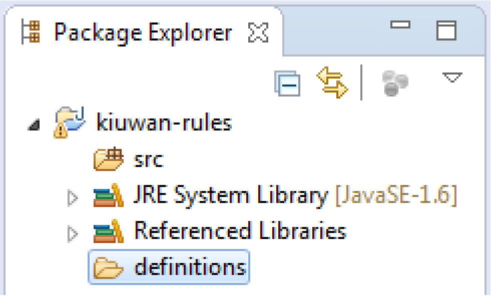
JARs and class folders on the build path:

```
analyzer.jar - C:\KiuwanLocalAnalyzer\lib
    Source attachment: (None)
    Javadoc location: file:/C:/KiuwanLocalAnalyzer/development/doc/javadoc/analyzer-javadoc.zip
    Native library location: (None)
    Access rules: (No restrictions)
spring.jar - C:\KiuwanLocalAnalyzer\lib
JRE System Library [JavaSE-1.6]
```

Add JARs... | Add Variable... | Add Library... | Add Class Folder... | Add External Class Folder... | Edit... | Remove | Migrate JAR File...

< Back Next > Finish Cancel

3.



Package Explorer

```
kiuwan-rules
    src
    JRE System Library [JavaSE-1.6]
    Referenced Libraries
    definitions
```

This is all that we need in the IDE part. Now we will launch Kiuwan Rule Developer and work simultaneously with both tools to create rules.

Step 2: Set up Kiuwan Rule Developer

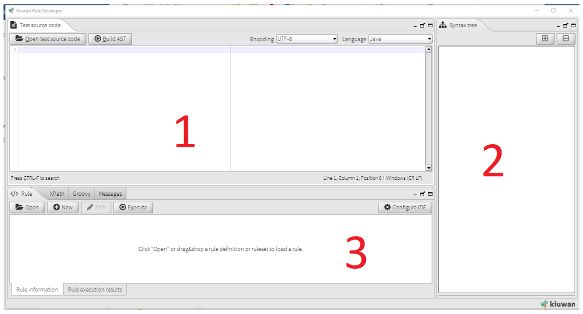
The Kiuwan Rule Developer is a GUI tool that will help you create, run and edit rules that can be executed in a Kiuwan analysis.

Once you have installed or upgraded Kiuwan Local Analyzer, follow these steps to run Kiuwan Rule Developer:

1. Run the software
    a. In Windows, double-click the script located in AGENT_HOME/development/rdev.cmd.
    b. In Unix OS, execute the AGENT_HOME/development/rdev.sh script,
2. Enter your Kiuwan account username and password.

The Kiuwan Rule Developer user interface is divided into three areas:

1. Top left: source code area: contains the source code the loaded rule will be run against.
2. Top right: AST (Abstract Syntax Tree) area: where the tree describing the parsed source code will be shown.
3. Bottom: the execution panels: where you can load, create and execute rules, XPath expressions, and groovy scripts.



Press CTRL-F to search

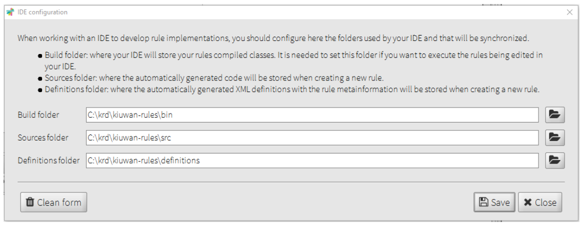
Line 1, Column 1, Position 0 | Windows (CR LF)

Click "Open" or drag&drop a rule definition or ruleset to load a rule.

Link the Kiuwan Rule Developer to your IDE

The first thing we will do is configure the Kiuwan Rule Developer to synchronize with an IDE. In this case, we will use the Eclipse project we have created before.

1. Click **Configure IDE** on the bottom right of Kiuwan Rule Developer
2. In the **Build Folder** field, type or look for the folder where Eclipse builds the project, in our case, */krd/Kiuwan-rules/bin*
3. In the **Sources Folder** field, type or look for the folder where Eclipse stores the source code, in our case, */krd/Kiuwan-rules/src*
4. In the **Definitions Folder** field, type or look for the folder we have created to store the rule definitions, in our case, */krd/Kiuwan-rules/definitions*



When working with an IDE to develop rule implementations, you should configure here the folders used by your IDE and that will be synchronized.

- Build folder: where your IDE will store your rules compiled classes. It is needed to set this folder if you want to execute the rules being edited in your IDE.
- Sources folder: where the automatically generated code will be stored when creating a new rule.
- Definitions folder: where the automatically generated XML definitions with the rule metainformation will be stored when creating a new rule.

Build folder C:\krd\kiuwan-rules\bin

Sources folder C:\krd\kiuwan-rules\src

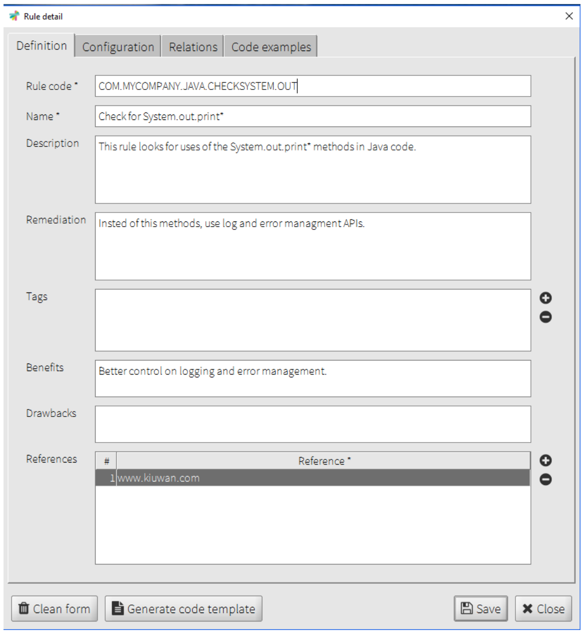
Definitions folder C:\krd\kiuwan-rules\definitions

Create a rule

Once the IDE and Kiuwan Rule Developer are wired, we can create a new rule definition.

1. Click **New**. The new rule window will appear.
2. We will create a rule to check the usage of System.out.print* methods in Java language, so we fill the fields conveniently.
    a. Fill the **Definition**, **Configuration** and **Relations** fields with values that fit the purpose of the rule.
    b. Fill the **Code examples** text areas with an example of the violation the rule will detect and how to fix it.
3. Once the fields are filled, click **Generate Code Template** to generate rule class implementation basic structure and then click **Save** to generate rule descriptor.

Step	Screenshot
2a.	



Rule code * COM.MYCOMPANY.JAVA.CHECKSYSTEM.OUT

Name * Check for System.out.print*

Description This rule looks for uses of the System.out.print* methods in Java code.

Remediation Insted of this methods, use log and error managment APIs.

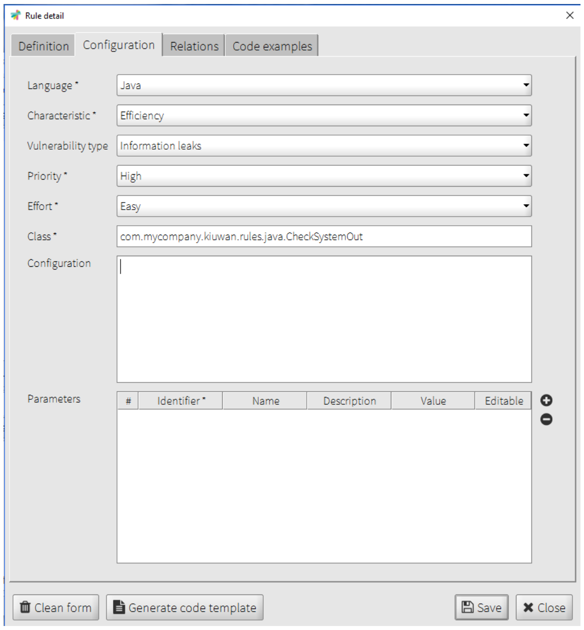
Tags

Benefits Better control on logging and error management.

Drawbacks

References

#	Reference *
1	www.kiuwan.com



Language * Java

Characteristic * Efficiency

Vulnerability type Information leaks

Priority * High

Effort * Easy

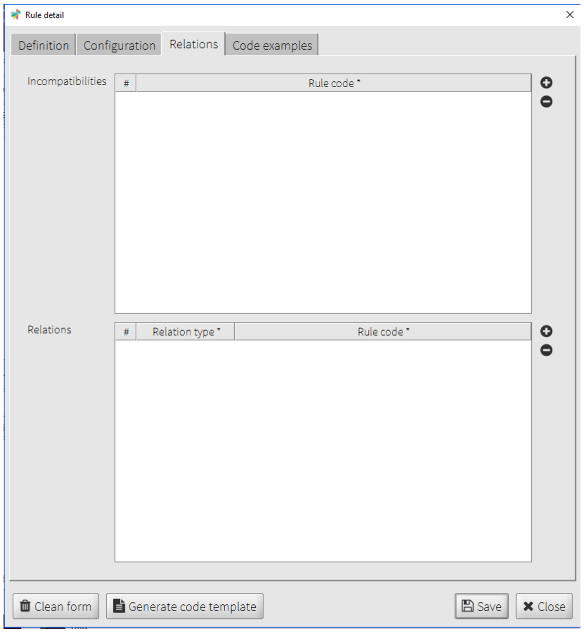
Class * com.mycompany.kiuwan.rules.java.CheckSystemOut

Configuration

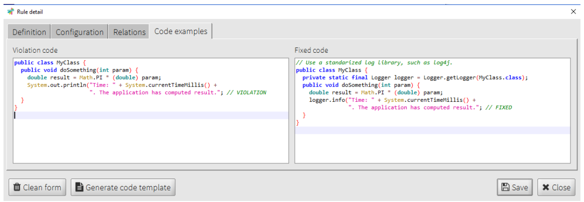
Parameters

#	Identifier *	Name	Description	Value	Editable

<table>
<tr><td></td><td>



</td></tr>
<tr><td>2b.</td><td>



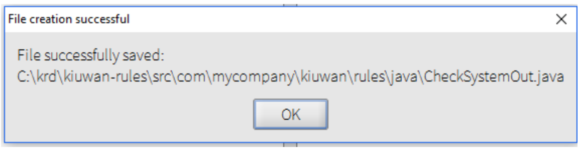
Violation code

```
public class MyClass {
  public void doSomething(int param) {
    double result = Math.PI * (double) param;
    System.out.println("Time: " + System.currentTimeMillis() +
                       ". The application has computed result."; // VIOLATION
  }
}
```

Fixed code

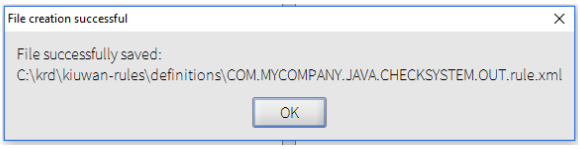
```
// Use a standarized log library, such as log4j.
public class MyClass {
  private static final Logger logger = Logger.getLogger(MyClass.class);
  public void doSomething(int param) {
    double result = Math.PI * (double) param;
    logger.info("Time: " + System.currentTimeMillis() +
                ". The application has computed result."; // FIXED
  }
}
```

</td></tr>
<tr><td>3.</td><td>



File creation successful

File successfully saved:
C:\krd\kiuwan-rules\src\com\mycompany\kiuwan\rules\java\CheckSystemOut.java



File creation successful

File successfully saved:
C:\krd\kiuwan-rules\definitions\COM.MYCOMPANY.JAVA.CHECKSYSTEM.OUT.rule.xml

</td></tr>
</table>

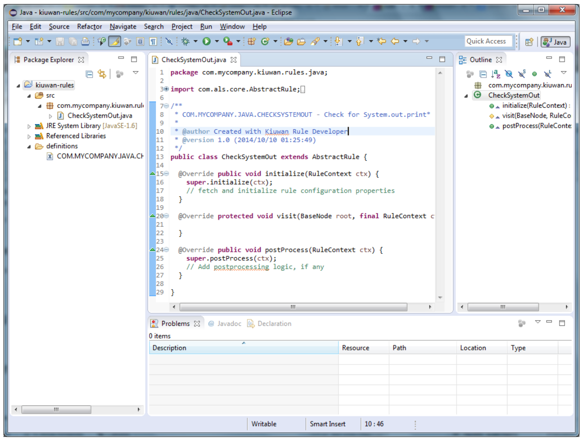
After saving the rule, two files will be created:

- The base source code of the rule, located in the previously configured build folder.
- The rule definition file, located in the previously configured definitions folder.

Once the new rule window is closed, Kiuwan Rule Developer will be prepared to execute your rule's implementation. But we first need to create that implementation.

Edit the rule template

Back to Eclipse, you should refresh to project contents to see the files created by Kiuwan Rule Developer.



```
package com.mycompany.kiuwan.rules.java;

import com.als.core.AbstractRule;

/**
 * COM.MYCOMPANY.JAVA.CHECKSYSTEMOUT - Check for System.out.print*
 *
 * @author Created with Kiuwan Rule Developer
 * @version 1.0 (2014/10/10 01:25:49)
 */
public class CheckSystemOut extends AbstractRule {

  @Override public void initialize(RuleContext ctx) {
    super.initialize(ctx);
    // fetch and initialize rule configuration properties
  }

  @Override protected void visit(BaseNode root, final RuleContext c

  }

  @Override public void postProcess(RuleContext ctx) {
    super.postProcess(ctx);
    // Add postprocessing logic, if any
  }

}
```

0 items

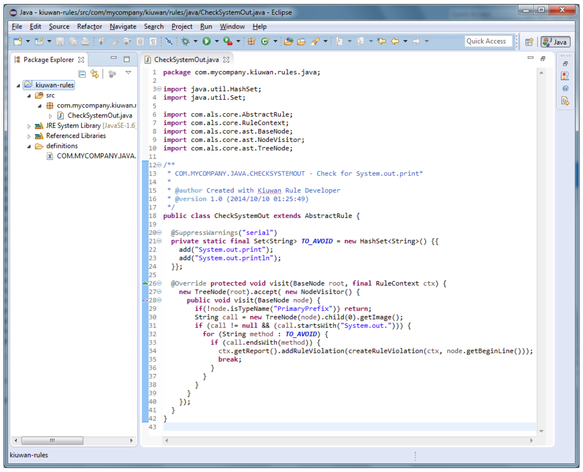
Description	Resource	Path	Location	Type

In our example, Kiuwan Rule Developer has created the following files:

- The **rule base code**: src/com/mycompany/Kiuwan/rules/java/CheckSystemOut.java
- The **rule definition**: definitions/COM.MYCOMPANY.JAVA.CHECKSYSTEMOUT.rule.xml

If you open the rule base code file, you will see the basic rule template. This is where you should add the code to detect the violations you want to find in your Java code. For more information on this topic, check the Getting Started with Rule Development guide.

In our example, we want to detect the uses of *System.out.print(String)* and *System.out.println(String)* in Java code. So this will be the source code for our rule:


Java - kiuwan-rules/src/com/mycompany/kiuwan/rules/java/CheckSystemOut.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
kiuwan-rules
src
com.mycompany.kiuwan.r
CheckSystemOut.java
JRE System Library [JavaSE-1.6]
Referenced Libraries
definitions
COM.MYCOMPANY.JAVA.
CheckSystemOut.java
kiuwan-rules

```
package com.mycompany.kiuwan.rules.java;

import java.util.HashSet;
import java.util.Set;

import com.als.core.AbstractRule;
import com.als.core.RuleContext;
import com.als.core.ast.BaseNode;
import com.als.core.ast.NodeVisitor;
import com.als.core.ast.TreeNode;

/**
 * COM.MYCOMPANY.JAVA.CHECKSYSTEMOUT - Check for System.out.print*
 *
 * @author Created with Kiuwan Rule Developer
 * @version 1.0 (2014/10/10 01:25:49)
 */
public class CheckSystemOut extends AbstractRule {

  @SuppressWarnings("serial")
  private static final Set<String> TO_AVOID = new HashSet<String>() {{
    add("System.out.print");
    add("System.out.println");
  }};

  @Override protected void visit(BaseNode root, final RuleContext ctx) {
    new TreeNode(root).accept( new NodeVisitor() {
      public void visit(BaseNode node) {
        if(!node.isTypeName("PrimaryPrefix")) return;
        String call = new TreeNode(node).child(0).getImage();
        if (call != null && (call.startsWith("System.out."))) {
          for (String method : TO_AVOID) {
            if (call.endsWith(method)) {
              ctx.getReport().addRuleViolation(createRuleViolation(ctx, node.getBeginLine()));
              break;
            }
          }
        }
      }
    });
  }
}

```

Eclipse compiles the rules as we save changes in the file (you should have the **Build automatically** option turned on, under the Project menu), so we already have a *.class* file in the build folder that Kiuwan Rule Developer can execute.

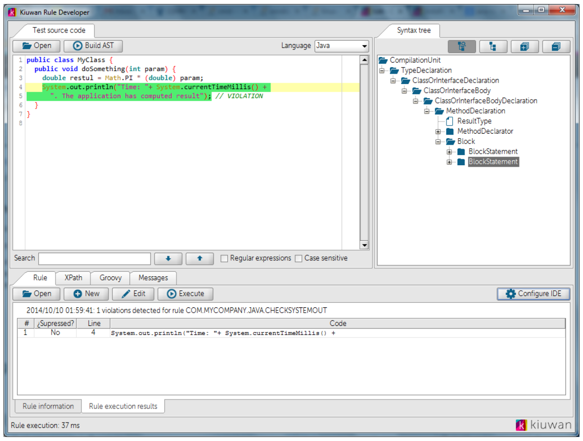
> **Kiuwan API Javadoc**
>
> Note that you can access the Kiuwan API Javadoc anytime from your IDE, as we configured the Javadoc Location of the analyzer.jar file when we created the New Java Project, in the "Preparing your IDE" section.

Execute the rule in Kiuwan Rule Developer

Switch back to Kiuwan Rule Developer.

We will need a source code to test the rule against. We will copy the code example we entered when we created the rule in the new rule dialog:

1. Copy or write down the source code in the **Test source code** tab.
2. Click **Generate AST** to parse the file and create an Abstract Syntax Tree.
3. Click **Execute** to see the results of applying the rule to the test code.



```
1 public class MyClass {
2   public void doSomething(int param) {
3     double restul = Math.PI * (double) param;
4     System.out.println("Time: "+ System.currentTimeMillis() +
5       ". The application has computed result"); // VIOLATION
6   }
7 }
8
```

- CompilationUnit
  - TypeDeclaration
    - ClassOrInterfaceDeclaration
      - ClassOrInterfaceBody
        - ClassOrInterfaceBodyDeclaration
          - MethodDeclaration
            - ResultType
            - MethodDeclarator
            - Block
              - BlockStatement
              - BlockStatement

2014/10/10 01:59:41: 1 violations detected for rule COM.MYCOMPANY.JAVA.CHECKSYSTEMOUT

#	¿Supressed?	Line	Code
1	No	4	`System.out.println("Time: "+ System.currentTimeMillis() +`

Rule execution: 37 ms

As you can see in the previous screenshot, the "Rule execution results" tab will be automatically selected with the results of applying the rule to the test source code. In our example, one violation will be detected in line 4. This line uses one of the methods the rule looks for.

You can modify the rule in your IDE and execute it in Kiuwan Rule Editor as many times as you need until you are satisfied with your rule's implementation. Kiuwan Rule Editor automatically detects changes in the compiled code, as long as you have correctly configured your IDE as we did in the previous sections.

Debug the rule in your IDE

Read more here: Debug custom rules

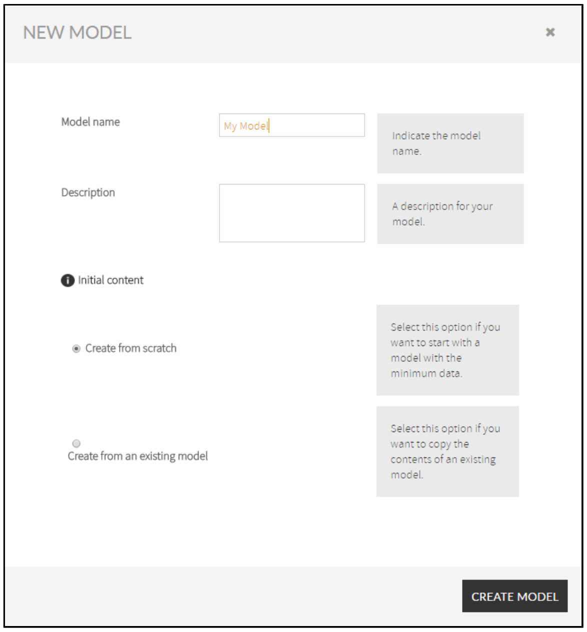
Step 3: Upload a Rule to Kiuwan Quality Models Manager

Once we are happy with our rule, the next step is to upload the rule definition to Kiuwan.

1. Access https://www.Kiuwan.com and log in to your Kiuwan account.
2. Go to **Model Management** module (click on the menu at the top right of the screen).
3. Create en empty quality model (New model, Create from scratch).
4. Go to the **Rules** section and click **Install rule definitions**.
    a. Click the "Upload" button and select the rule definition that Kiuwan Rule Developer has created in the definitions folder.
    b. Click "Next" and click on the quality model you have just created to add your rule to it.
    c. Click on the "Install" button to proceed.
5. Your rule will be installed into your library and will be added to the new quality model.
6. Publish the quality model so you can assign it to one of your applications for analysis.
7. Click on the configuration icon (top right of the screen) and click on the Application management option.
    a. Select the application you will analyze with the new quality model that contains your custom rule.
    b. Open the drop-down menu in the Quality parameters section and select the quality model you have created in step 3.

Step	Screenshot

3.



NEW MODEL

Model name: My Model

Indicate the model name.

Description:

A description for your model.

Initial content

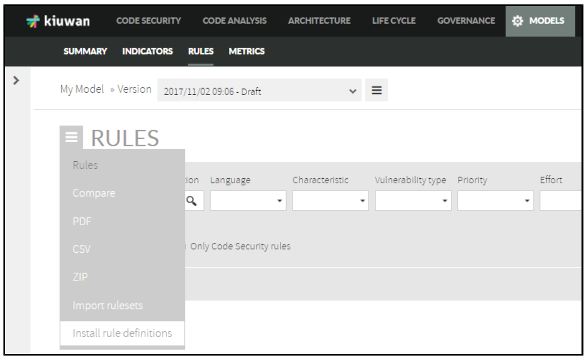
● Create from scratch

Select this option if you want to start with a model with the minimum data.

○ Create from an existing model

Select this option if you want to copy the contents of an existing model.

4.

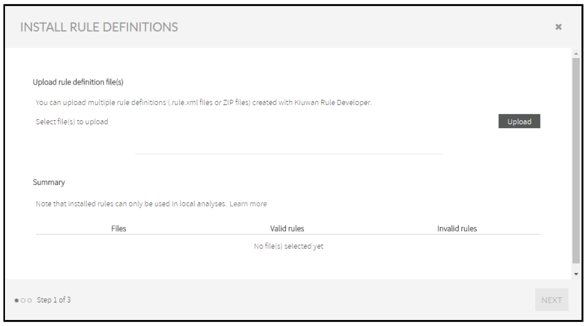


My Model » Version 2017/11/02 09:06 - Draft

RULES

Only Code Security rules

4.a.



INSTALL RULE DEFINITIONS

Upload rule definition file(s)

You can upload multiple rule definitions (.rule.xml files or ZIP files) created with Kiuwan Rule Developer.

Select file(s) to upload

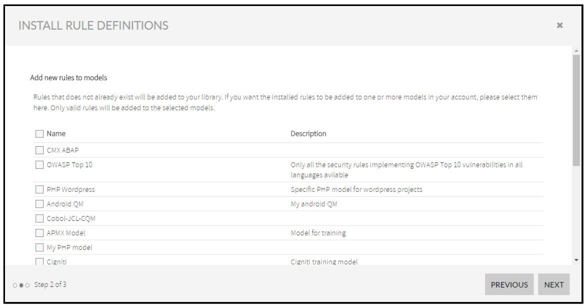
Summary

Note that installed rules can only be used in local analyses. Learn more

Files	Valid rules	Invalid rules
No file(s) selected yet		

● ○ ○ Step 1 of 3

4.b.



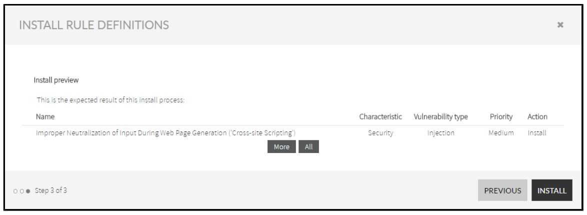
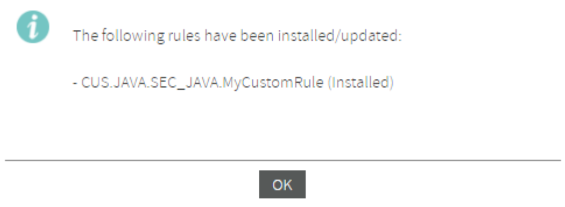
INSTALL RULE DEFINITIONS

Add new rules to models

Rules that does not already exist will be added to your library. If you want the installed rules to be added to one or more models in your account, please select them here. Only valid rules will be added to the selected models.

□ Name	Description
□ CMX ABAP	
□ OWASP Top 10	Only all the security rules implementing OWASP Top 10 vulnerabilities in all languages avilable
□ PHP Wordpress	Specific PHP model for wordpress projects
□ Android QM	My android QM
□ Cobol-JCL-CQM	
□ APMX Model	Model for training
□ My PHP model	
□ Cigniti	Cigniti training model

○ ● ○ Step 2 of 3

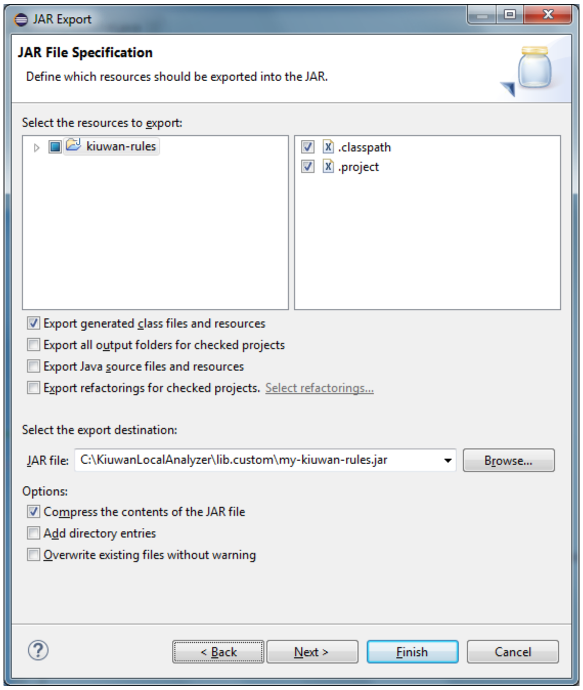
4.c.	 INSTALL RULE DEFINITIONS Install preview This is the expected result of this install process: Name / Characteristic / Vulnerability type / Priority / Action Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') / Security / Injection / Medium / Install More All ○ ○ ● Step 3 of 3 PREVIOUS INSTALL
5.	 The following rules have been installed/updated: - CUS.JAVA.SEC_JAVA.MyCustomRule (Installed) OK

After following the previous steps, everything should be prepared to run a new analysis with Kiuwan Local Analyzer.

Step 4: Execute a custom rule analysis with Kiuwan Local Analyzer

Once your application has assigned the new quality model, we have to create a package with our rule so Kiuwan Local Analyzer can execute it.

1. In Eclipse, right-click on the **rules project** and choose **Export... Java** > **JAR File**.
2. Browse for the lib.custom folder under the Kiuwan Local Analyzer folder and write a name for your rules jar archive. For example AGENT_HOME/lib.custom/my-Kiuwan-rules.jar
3. Click **Finish** to export your rules project.


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
kiuwan-rules
☑ .classpath
☑ .project
☑ Export generated class files and resources
□ Export all output folders for checked projects
□ Export Java source files and resources
□ Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: C:\KiuwanLocalAnalyzer\lib.custom\my-kiuwan-rules.jar
Browse...
Options:
☑ Compress the contents of the JAR file
□ Add directory entries
□ Overwrite existing files without warning
< Back
Next >
Finish
Cancel

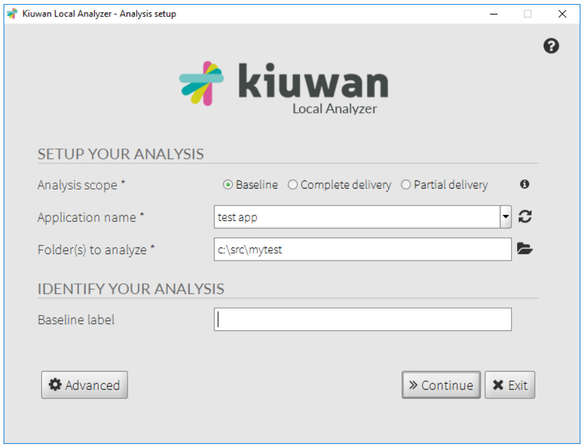
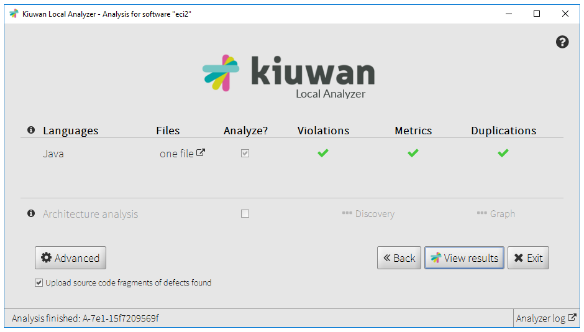
Now everything should be prepared to launch your analysis.

The next step is to run Kiuwan Local Analyzer and analyze your application. If you need more information on this topic, please read the Kiuwan Local Analyzer page.

We will use a test application made of two Java classes. One with a rule violation, one with no violations. We will call this application **Test application**.

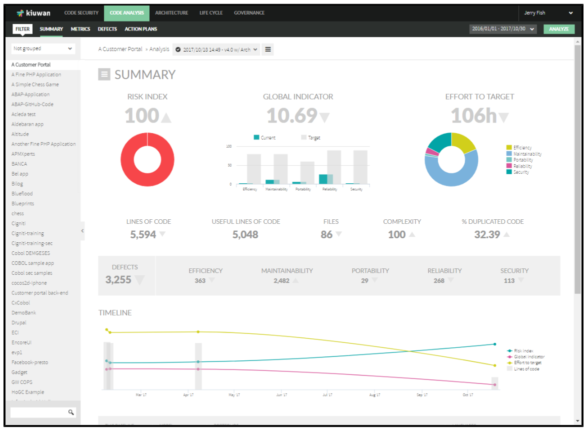
Follow these steps to analyze the application in Kiuwan Local Analyzer:

1. Run Kiuwan Local Analyzer by double clicking *AGENT_HOME/Kiuwan.cmd* script in Windows or by running the *AGENT_HOME/Kiuwan.sh* script in Unix OS.
2. Log in with your Kiuwan account.
3. Select the application you want to analyze with your custom rules. In our case, this will be **Test application**.
4. Select the source code you want to analyze and click **Next**.
5. Click **Analyze**. You will see in the status bar (bottom of the window) that Kiuwan Local Analyzer first downloads the quality model associated with the selected application, in our example, **My company quality model** and, then, analyzes the source code you have previously configured.
6. Click **View results** to access the Kiuwan site and see the results.

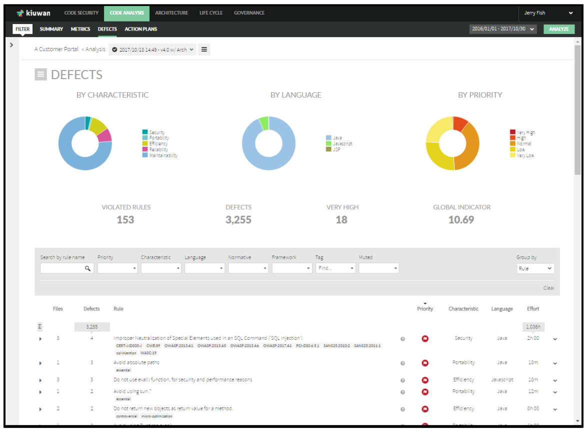
Step	Screenshot
3. and 4.	 Kiuwan Local Analyzer - Analysis setup SETUP YOUR ANALYSIS Analysis scope * ● Baseline ○ Complete delivery ○ Partial delivery Application name * test app Folder(s) to analyze * c:\src\mytest IDENTIFY YOUR ANALYSIS Baseline label
6.	 Kiuwan Local Analyzer - Analysis for software "eci2" Languages | Files | Analyze? | Violations | Metrics | Duplications Java | one file | ☑ | ✓ | ✓ | ✓ Architecture analysis ☐ Discovery Graph ☑ Upload source code fragments of defects found Analysis finished: A-7e1-15f7209569f kiuwan Local Analyzer Advanced Back View results Exit Analyzer log

Step 5: View the Analysis Results

Once the analisis is done, you can check the results by clicking **View results**. A browser window will open and the Kiuwan web page will appear with the results of the analysis.



Access the **Defects** section to see a defect emitted by the custom rule you have just created with Kiuwan Rule Developer.



Files	Defects	Rule	Priority	Characteristic	Language	Effort
Σ	3,255					1,036h
3	4	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') CERT-J:IDS00-J CWE:89 OWASP:2013:A1 OWASP:2013:A5 OWASP:2013:A6 OWASP:2017:A1 PCI-DSS:6.5.1 SANS25:2010:2 SANS25:2011:1 sql-injection WASC:19		Security	Java	2h 00
1	3	Avoid absolute paths essential		Portability	Java	18m
3	3	Do not use eval() function, for security and performance reasons		Efficiency	Javascript	18m
1	2	Avoid using sun.* essential		Portability	Java	12m
2	2	Do not return new objects as return value for a method. controversial micro-optimization		Efficiency	Java	8h 00

Summary

In this quick start guide, we have seen how to create, modify, pack, install and execute a custom rule for Kiuwan. We can summarize the needed steps in the following list:

1. **IDE**: create a project and configure it to be used with Kiuwan Rule Editor.
2. **Kiuwan Rule Developer**: wire with your IDE.
3. **Kiuwan Rule Developer**: create a custom Kiuwan rule.
4. **IDE**: implement your custom rule from the automatically created template.
5. **Kiuwan Rule Developer**: execute and check the results of your custom rule.
6. **Kiuwan**: install your custom rule and use it in a quality model.
7. **IDE**: pack your custom rule code and deploy it to Kiuwan Local Analyzer.
8. **Kiuwan Local Analyzer**: execute an analysis of your application.
9. **Kiuwan**: view the results of analyzing your application with your custom rule.

Next Steps

The next steps you should follow are:

- Read the Getting Started with Rule Development guide.
- Read the Rule Development Manual, found under the AGENT_HOME/development/doc folder.
- Check the rule examples included under the AGENT_HOME/development/doc/samples folder.