Configuring an external plugin to enrich the results for your Universal Analyzer language

If the language you are targeting in the Universal Analyzer has a complementary third party tool that can analyze source code and generate Quality Rule data, then you can integrate this tool into the Language Package as a "plugin" and execute it during the Universal Analyzer analysis. This section describes the process for configuring a plugin and how to integrate it into your language pack.

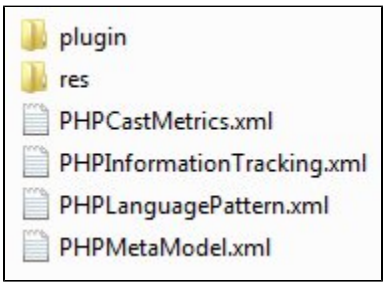
Introduction

Languages not already covered by CAST AI¨P (via a dedicated CAST Analyzer) can be analyzed using the Universal Analyzer and a custom Language Package specifically for your language. If the technology that you are targeting has a third party tool or application that can provide Quality Rule data as a complement to the source code analysis data generated during the analysis, then you can integrate this tool into the Language Package as a "plugin" and automate its execution. The tool(s) in the plugin will be executed **after the analysis** (if a snapshot is generated, which includes an analysis, then the plugin will be executed after the analysis) and data will be displayed in the Engineering Dashboard/Health Dashboard as with any other language supported as standard by CAST AIP.

Creating a plugin

A plugin consists of a folder ("**plugin**") containing a batch file ("**launch.bat**") containing the command line instructions required to execute the third party tool. The folder must be placed in your language package at the same level as the xxxMetaModel.xml file and other XML files.

Take the following example for PHP:



You also need to include in the **plugin** folder **any other batch files** or **tools** that are required.

launch.bat

The **launch.bat** file will contain the instructions for running your third party tool - what you put in this file entirely depends on how the tool is run. Alongside the launch.bat file, you can include any other batch files, tools (such as .exe or .jar) or tools/batch files in sub-folders that need to be run - they will be called from your launch.bat.

The script and tools executed in the launch.bat file must format the output data in a specific way so that the Universal Analyzer can assign the data to the objects that were created during the analysis phase (run prior to the execution of the launch.bat file). The Universal Analyzer will expect the output data in the form of an **XML file** called **result.xml** (see below), located at the same level as the **launch.bat** file in the **plugin** folder.

How it works

After creating a version with the CAST Delivery Manager Tool containing your source code and having correctly packaged/installed your custom Universal Analyzer language package so that this source code is identified correctly, when you launch an analysis or a snapshot generation (including an analysis) with the CAST Management Studio, CAST will first check to see whether the language pack contains a folder called "**plugin**" containing a file called "**launch.bat**" - if these two criteria are met, then the launch.bat file will be executed on completion of the analysis, using one parameter:

- the input folder (i.e. where your analyzed source is stored - you do not need to configure this, it is managed entirely by the CAST Management Studio).

In a situation where the source code is organized into **multiple root level folders without parents**, each folder path will be passed to the launch.bat script successively.

If the language pack does not contain a "**plugin**" folder containing a "**launch.bat**", then the process continues as normal.

Output to result.xml

The Universal Analyzer will expect the output of your launch.bat script and tools in a file called **result.xml** located at the same level as the **launch.bat** file in the **plugin** folder.

This file should be formatted as follows:

- A top level node called "**plugin**"
- As many child nodes called "**file**" as there are source files passed to the launch.bat script via the source code folder(s). These nodes must have an attribute called "**name**" that indicates the full path to the analyzed file.
- Each child node "file" can then contain as many child nodes called "**violation**" that the plugin scripts/tools have returned as a Quality Rule violation. Each of these "violation" nodes can have multiple attributes, but the Universal Analyzer requires only two:
    - "**rule**" - indicates the name of the Quality Rule that is violated by an object in the parent file
    - "**beginline**" - indicates the line number on which the object violating the Quality Rule starts - this is used to map identify the object in violation.

For example:

```
<?xml version="1.0" encoding="UTF-8" ?>

<plugin>

        <file name="E:\COE\Jira_231\SOURCE\PHP\TestCases\ActiveDynamicPage.php" errors="60" warnings="0">

                <violation beginline="28" column="5" rule="Squiz.Commenting.ClosingDeclarationComment.Missing"
severity="5">Expected //end main()</violation>

                <violation beginline="37" column="5" rule="Squiz.Commenting.ClosingDeclarationComment.Missing"
severity="5">Expected //end main()</violation>

                <violation beginline="42" column="5" rule="Squiz.Commenting.ClosingDeclarationComment.Missing"
severity="5">Expected //end bar()</violation>

        </file>

         <file name="E:\COE\Jira_231\SOURCE\PHP\TestCases\unusedFun.php">

                <violation beginline="5" endline="5" rule="UnusedPrivateMethod" ruleset="Unused Code Rules"
package="+global" externalInfoUrl="http://phpmd.org/rules/unusedcode.html#unusedprivatemethod" class="
Something" method="foo" priority="3">Avoid unused private methods such as 'foo'.</violation>

        </file>

</plugin>
```

Tips and remarks

The following tips and remarks may prove useful:

- The "rule" attribute in the "violation" node must use the same format as the properties defined on objects resulting from the analysis as defined in the XXXMetaModel.xml file.
- If the Universal Analyzer cannot locate objects resulting from its analysis using the "beginline" attribute, then the violation will be mapped to a **sourceFile** type object (see Defining a new language). As such, you must configure the XXXMetaModel.xml file in such a way that all the properties returned by the plugin tools/scripts can be assigned to a **sourceFile** type object.
- If a file is taken into account by your plugin but has not been configured in the Language Package, then the results output by the plugin will be ignored.