PHP 3.1

Summary: This document provides information about the extension providing **PHP** support.

Extension ID

com.castsoftware.php

What's new?

Please see **PHP 3.1 - Release Notes** for more information.

Description

This extension provides support for applications written using the **PHP** language.

In what situation should you install this extension?

If your application contains source code written using **PHP** and you want to view these object types and their links with other objects, then you should install this extension.

Supported Versions of PHP

Although this extension is officially supported by CAST, please note that it has been developed within the technical constraints of the CAST Universal Analyzer technology and to some extent adapted to meet specific customer needs. Therefore the extension may not address all of the coding techniques and patterns that exist for the target technology and may not produce the same level of analysis and precision regarding e.g. quality measurement and/or function point counts that are typically produced by other CAST AIP analyzers.

This version of the extension provides support for:

PHP version	Supported
5.x	✓

Function Point, Quality and Sizing support

This extension provides the following support:

- **Function Points (transactions)**: a green tick indicates that OMG Function Point counting and Transaction Risk Index are supported
- **Quality and Sizing**: a green tick indicates that CAST can measure size and that a minimum set of Quality Rules exist

Function Points (transactions)	✓
Quality and Sizing	✓

AIP Core compatibility

This extension is compatible with:

AIP Core release	Supported?
8.3.x	✓

Supported DBMS servers

DBMS	Supported?
CSS / PostgreSQL	✓

Download and installation instructions

The extension will be automatically downloaded and installed in AIP Console when you deliver PHP code. You can also **manually install** the extension using the **Application - Extensions** interface. When installed, follow the instructions below to run a new analysis/snapshot to generate new results:

- **Advanced onboarding - run and validate the initial analysis**
- **Advanced onboarding - snapshot generation and validation**

Once the extension is downloaded and installed, you can now package your source code and run an analysis. The process of preparing and delivering your source code is described below.

Upgrading from previous releases of the PHP Analyzer

Previous releases of the PHP Analyzer required that **an instance of PHP** was installed on the AIP Node (i.e. the machine on which the analysis is run). This requirement has been removed in **3.1.0.** The PHP Analyzer now uses an instance of PHP embedded within the extension. Therefore, please note the following:

- If you have already installed **a previous version** of the PHP Analyzer on your AIP Node and already have a functioning PHP install from that extension, please ensure that you **uninstall PHP** before proceeding with the instructions below. To remove the PHP installation provided with the PHP Analyzer:
  - **delete the folder** into which it was installed (by default this is usually set to **C:\php**).
  - **delete** the system environment variable **PHP_HOME**
- Please check that you **do not** have an existing third party (i.e. not provided by CAST) installation of PHP on this machine. If a third party installation of PHP already exists, please follow the PHP uninstall procedure for the install method that was used, before starting an analysis. Third party PHP installations are **not compatible** with the PHP extension.

Prepare and deliver the source code

Source code preparation

- Only files with following extensions will be analyzed ***.php; *.php4; *.php5; *.php6; *.inc; *.phtml.** The ***.yml** and ***.yaml** extensions are also supported for Symfony framework.
- The analysis of **XML** and **XSL** files contained in the PHP application is not supported.
- The analysis of any **HTML** and **JavaScript** source code delivered with the PHP code is managed by the HTML and JavaScript extension / .NET analyzer, to be configured in addition to the PHP analysis.

Source code preprocessing

PHP source code needs to be preprocessed so that CAST can understand it and analyze it correctly. This code preprocessing is **actioned automatically** when an analysis is launched or a snapshot is generated (the code is preprocessed before the analysis starts). In other words you only need to package, deliver and launch an analysis/generate a snapshot for the preprocessing to be completed. The PHP Preprocessor log file is stored in the following location:

```
%PROGRAMDATA%\CAST\CAST\Logs\<application_name>\Execute_Analysis_<guid>\com.castsoftware.php.
<_extension_version>.prepro_YYYYMMDDHHMMSS.log
```

Note that the LISA folder will be used to analyze the preprocessed files.

Short tags

PHP short tags **<?** and **<?=** in the delivered source code cannot be handled as is, therefor the analyzer will automatically convert them to <**?php** tags with an added comment, for example: **<?=$string?>** will be transformed into <**?php /*php short tag*/echo $string>**.

Deliver the source code

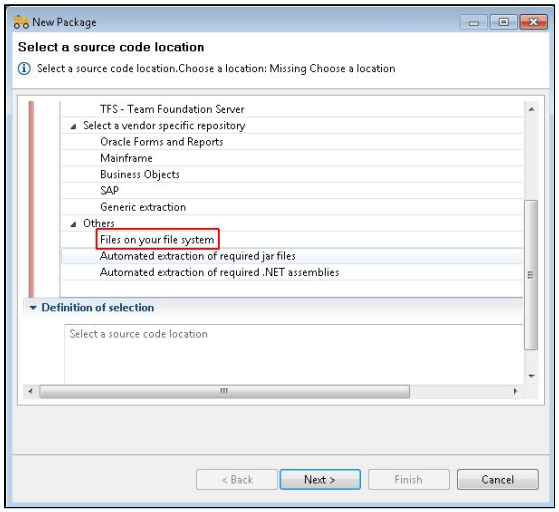
Using AIP Console

- **Advanced onboarding - add a new version and deliver source code**
- **Advanced onboarding - validate and accept the version**

Using CAST Management Studio

Create a new **Version** - the **Delivery Manager Tool** will open. Then create a new **Package** for your source code using the **Files on your file system** option and choose the location of your source code:

Click to enlarge



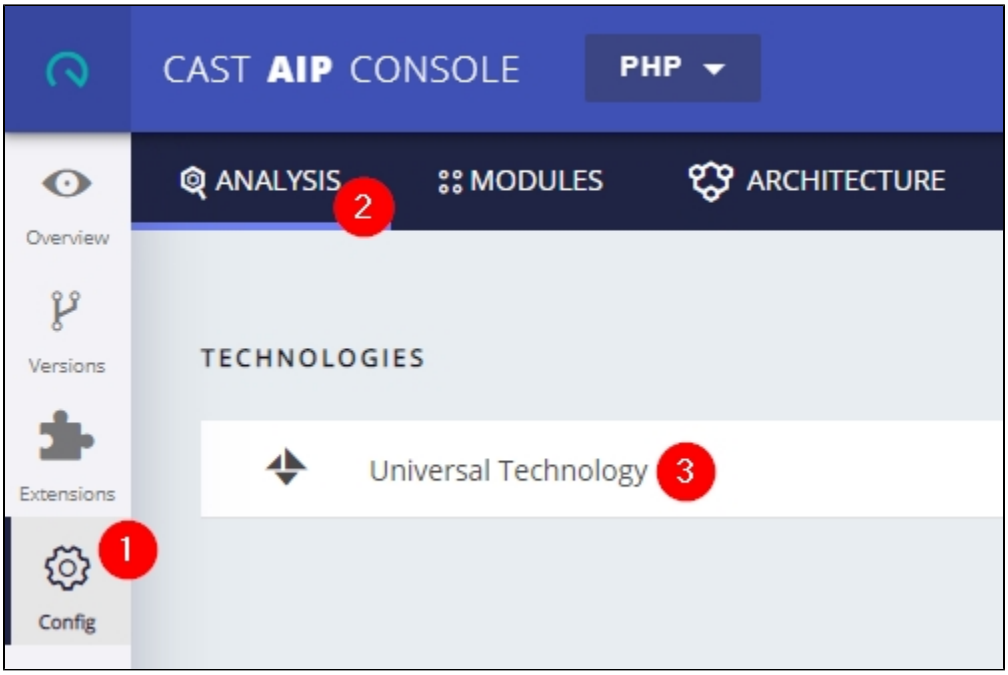
Run the **Package action** and check the **packaging results** before delivering the source code and accepting the version in CAST Management Studio.

Analysis configuration and execution

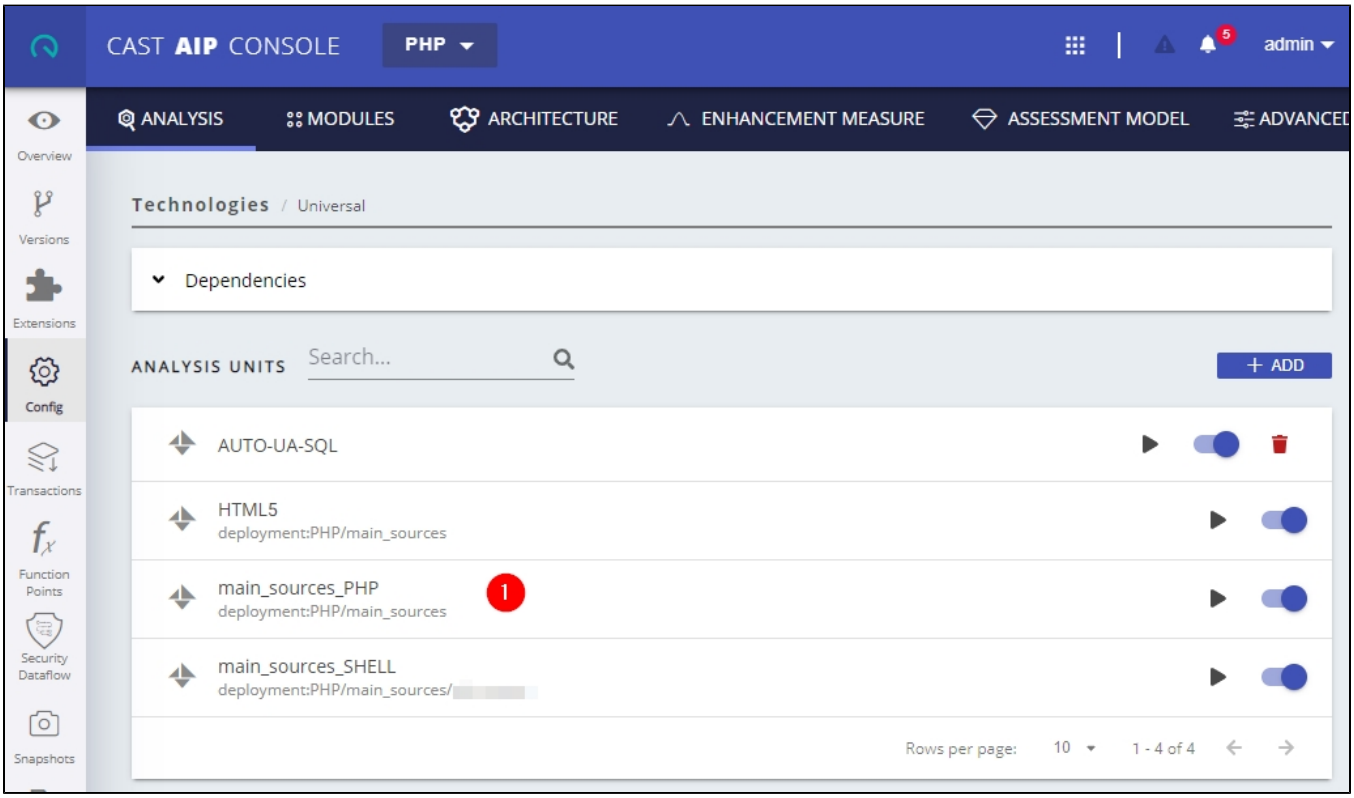
Using AIP Console

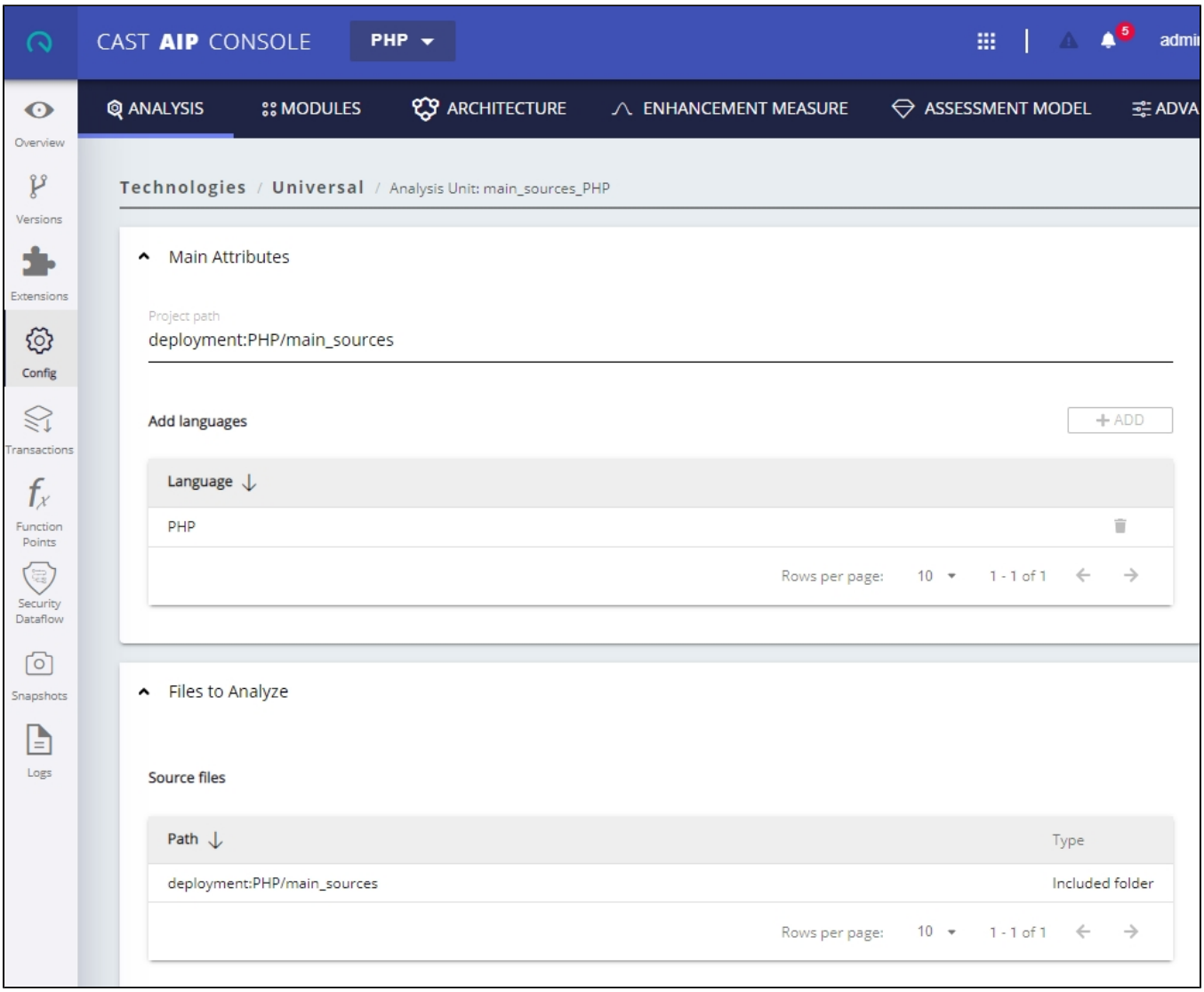
There are **no analysis/technology configuration options** available for PHP, however you should check that at least one PHP analysis unit has been created as shown below.

AIP Console exposes the technology configuration options once a version has been **accepted/imported**, or an **analysis has been run**. Click **Universal Technology (3)** in the **Config (1)** > **Analysis (2)** tab to display the available options for your PHP source code:



Then choose the relevant **Analysis Unit (1)** to view the configuration:

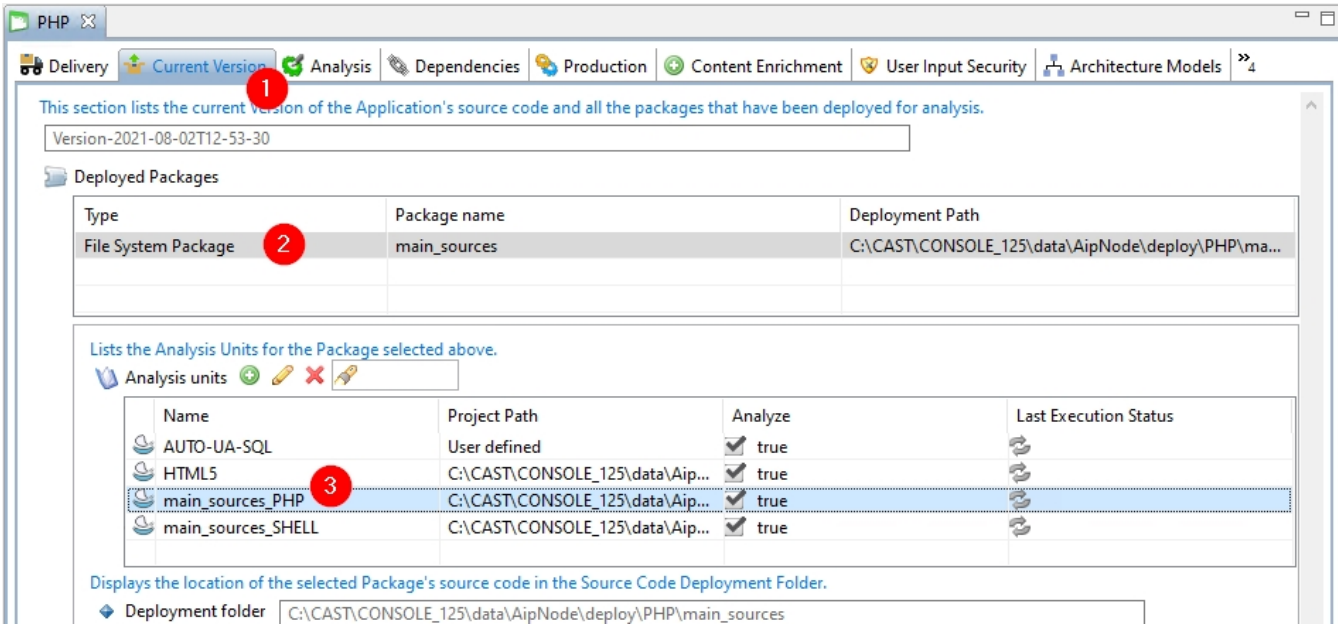


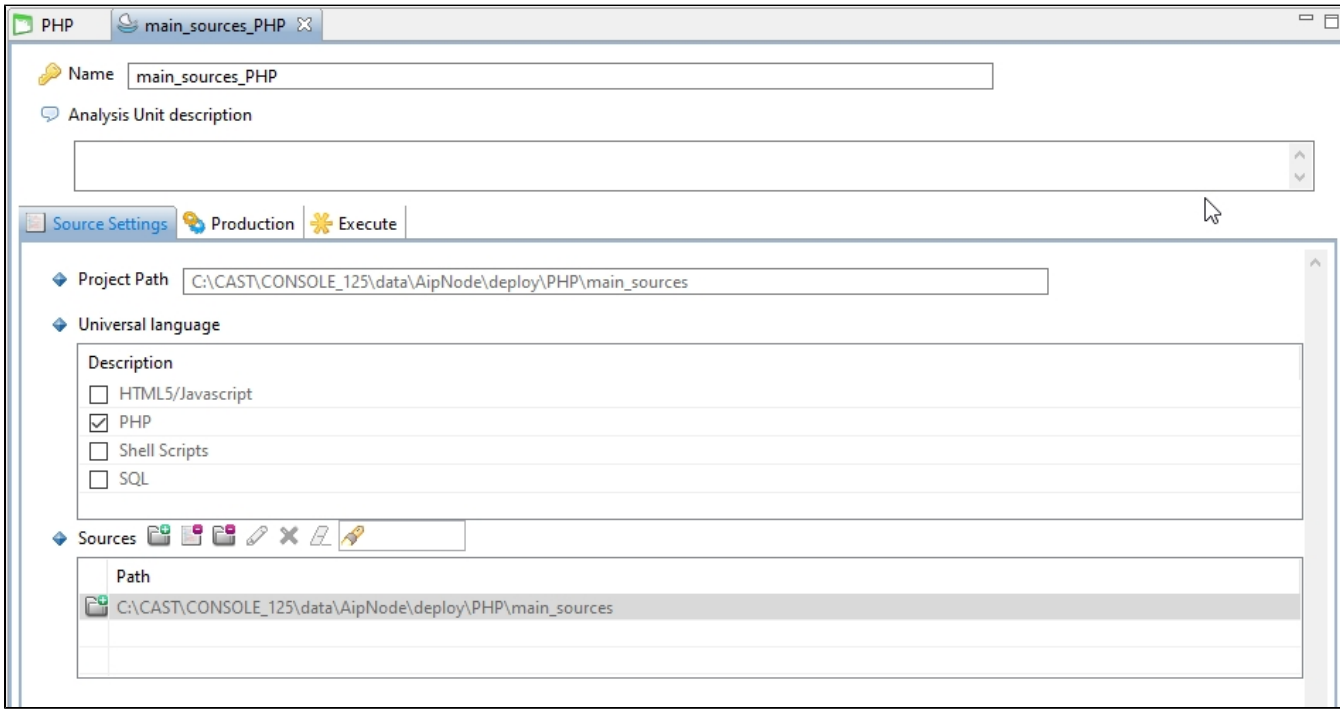


Using CAST Management Studio

> There are **no analysis/technology configuration options** available for PHP, however you should check that at least one PHP analysis unit has been created as shown below.

In the **Current Version** tab (1) select the **deployed package** (2) and ensure that at least one **Analysis Unit** (3) has been created for your PHP source code:





What results can you expect?

Objects

PHP Objects

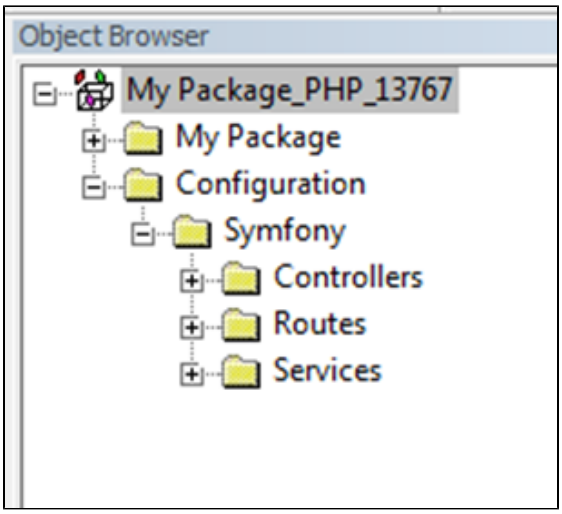
Icon	Metamodel Name
	PHP Array

	PHP Class
	PHP Class Constant
	PHP Constructor
	PHP Define
	PHP Function
	PHP Interface
	PHP Member
	PHP Method
	PHP Section
	Script Function
	Script Section

Symfony Framework objects

Icon	Metamodel Name
	PHP Symfony Controller
	PHP Symfony Controller Class
	PHP Symfony Route
	PHP Symfony Service

In CAST Enlighten, all Symfony objects will appear under their respective folders as shown below :



PHP Symfony Controller Class

- **Supported scenario**: If the Class name ends with Controller, we will create PHP Symfony Controller Class objects
- **Links**:
  - PHP Symfony Controller Class --- Refer Link ---> PHP Class
- **Limitations**: Alternate syntax where you can give the class name that does not have suffix "Controller" is not supported

PHP Symfony Controller

- **Supported scenario**: If the method or function ends with suffix "Action", then PHP Symfony Controller Object will be created
- **Links**:
  - PHP Symfony Controller --- Refer Link ---> PHP Symfony Route
  - PHP Symfony Controller --- Refer Link ---> PHP Method\Function

PHP Symfony Route

- **Supported scenario**:
  - If a route has been declared in the yml file, a route object will be created
  - If a route has been declared in PHP file an annotation route object will be created as follows:
    - Default naming convention for route annotation when declared without name above class "<classname>_Class_Annotation_<number>"
    - Default naming convention for route annotation when declared without name above method "<methodname>_Method_Annotation_<number>"
- **Links**:
  - PHP Symfony Route --- Call Link ---> PHP Symfony Controller

PHP Symfony Service

- **Supported scenario**: If a service has been declared in the yml configuration files, PHP Symfony Service Object will be created
- **Links**:
  - PHP Symfony Service --- Call Link ---> PHP Method
  - PHP Symfony Service --- Call Link ---> PHP Property
  - PHP Symfony Service --- Call Link ---> PHP Class constructor
- **Limitation**: Inheritance is not supported while determining property setter or constructor injection - they need to be defined in the same class which is being referred to in the service

Structural Rules

The following structural rules are provided:

3.1.1-funcrel	https://technologies.castsoftware.com/rules?sec=srs_php&ref=||3.1.1-funcrel
3.1.0-beta1	https://technologies.castsoftware.com/rules?sec=srs_php&ref=||3.1.0-beta1

You can also find a global list here:

https://technologies.castsoftware.com/rules?sec=t_1017000&ref=||

Logging mechanism

Analysis log files

Analysis logs are stored in the default locations.

PHP Preprocessor

PHP Preprocessor log file name (the preprocessor is launched automatically during an analysis) would be in format com.castsoftware.php.prepro_<ExtensionVersion>_<YYYYMMDDHHMMSS>.log

PHP Plugin

PHP Plugin, which uses PHP CodeSniffer, PDepend, PMD, log file name (the PHP Plugin is launched automatically during an analysis) would be of format com.castsoftware.php.plugin_<ExtensionVersion>_<YYYYMMDDHHMMSS>.log.

Errors and Warnings

The PHP configuration included in the extension uses external plugins. During the analysis, the Universal Analyzer or the plugin can throw errors or warnings. The table below list the most significant errors/warnings and lists a suggested remediation action:

Tool	Error or Warning	Action
Analyzer & Code Sniffer	UA Plugin : No property (.....) found in meta model for php...	No action required. The analyzer is telling you that not all the properties are considered to be injected into the Analysis Service.

Limitations

LISA path length limited to 256 characters

If the LISA (Large Intermediate Storage Area) path for a specific file exceeds 256 characters, violation calculation for this file will fail with message "<filepath> does not exist". This warning will appear in com.castsoftware.plugin*.log file. This is a limitation of PHP itself and not the PHP extension. To remediate this issue reduce path to the LISA folder where possible.

Name matching links - Universal Analyzer limitation

Due to a limitation in the Universal Analyzer (the "engine" used for PHP analyses), links will be created from any name to any matching name. At a minimum the following rule may be impacted and give erroneous results:

1007004	Avoid Methods and Functions with High Fan-In (PHP)
1007006	Avoid Methods and Functions with High Fan-Out (PHP)
1007008	Avoid JavaScript Functions with High Fan-In (PHP)
1007010	Avoid JavaScript Functions with High Fan-Out (PHP)
1007168	Avoid using function or method return value that do not have return (PHP)
1007170	Avoid function return value ignored (PHP)

Analysis of XML and XSL files contained in the PHP application

The analysis of XML and XSL files contained in the PHP application is not supported. Any links between these files and any other file in the application will not be detected. This will impact the results of all the Quality Rules using these files.

Support of JavaScript source code

The PHP extension does not support JavaScript and as such, any JavaScript source code located in PHP or JavaScript files will not be analyzed. CAST recommends using the HTML5 and JavaScript extension to analyze JavaScript files in the source code.

Support of PHTML files

PHTML files are supported with some limitations. If the files contain calls to functions or methods defined in other files and these other files are not specifically included, then these links will be lost.

Missing Links

If a php class has members declared on the same line, only the first member will be detected. For example:

```
class Test {

public $first, $second, $third;

}
```

After analysis only object for "first" will be created.

Limitations specific to rules

Avoid artifacts having recursive calls

"Avoid artifacts having recursive calls" (7388 - a standard CAST rule) - in some cases, a false positive may be detected: a call to a parent function can be detected as a recursive call

> Note that an equivalent rule specific to the PHP extension (**Avoid artifacts having recursive calls (PHP) - 1007242**) was added in PHP 1.2.0. This replacement rule now produces accurate results and the results of 7388 should be ignored.

Avoid using break or continue statements in loops with high cyclomatic complexity

"Avoid using break or continue statements in loops with high cyclomatic complexity" (1007176) - if the break statement is located in JavaScript functions, no violations will be detected. JavaScript source code located in .PHP or JavaScript files is not analyzed (see limitation listed above).

Avoid unreferenced PHP Files

The rule "Avoid unreferenced PHP Files" (1007052) will return a false positive violation when a PHP file is referenced **only** from other technologies, for example from only within html/javascript source code.

License agreements

The PHP extension uses several third-party tools. The Licence Agreements for these tools are listed below:

PHP_CodeSniffer

More information about this tool is available here: http://pear.php.net/package/PHP_CodeSniffer

Version

CAST ships version **2.5.0** of the PHP_CodeSniffer.

License

The licence agreement for the PHP_CodeSniffer tool is available here:

- https://opensource.org/licenses/BSD-2-Clause

and is detailed below:

Copyright (c) 2012, Squiz Pty Ltd (ABN 77 084 670 600)
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- *Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.*
- *Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.*
- *Neither the name of Squiz Pty Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.*

Quality Rules calculated by the PHP_CodeSniffer tool

Rule name	ID
Avoid artifacts using "for" loops which can be simplified to a "while" loop (PHP)	1007022
Avoid incrementer jumbling in loops (PHP)	1007024
Use identical type operator rather than "equal" operator (PHP)	1007026
Use increment/decrement operators where possible (PHP)	1007028
Avoid using empty statement (PHP)	1007030
Avoid empty class definition (PHP)	1007032
Avoid classes having excessive number of derived classes(PHP)	1007036
Avoid classes having excessive number of dependencies (PHP)	1007038
Avoid Classes with High Depth of Inheritance Tree (PHP)	1007046
Avoid unnecessary final modifiers inside final Classes (PHP)	1007056
Avoid unused parameters (PHP)	1007058
Avoid Class name not matching parent file name (PHP)	1007080
Use lowercase for control structures in Sections (PHP)	1007084
Use lowercase for control structures in Methods and Functions (PHP)	1007086
Avoid having variable with too short name (PHP)	1007088
Avoid having variable with too long name (PHP)	1007090
Avoid "elseif" statements (PHP)	1007096
Avoid Functions throwing exceptions and not having a @Throws tag (PHP)	1007124
Avoid classes exceeding maximum length (PHP)	1007126
Avoid methods having too many parameters (PHP)	1007128
Avoid Methods exceeding maximum length (PHP)	1007130
Avoid classes with too many fields (PHP)	1007132
Avoid classes with too many methods (PHP)	1007134
Avoid classes having a number of public methods and attributes exceeds maximum (PHP)	1007136
Avoid having unused variables (PHP)	1007138
Avoid unused private fields (PHP)	1007140
Avoid unused private methods (PHP)	1007142
Avoid classes exceeding number of weighted methods (PHP)	1007144
Avoid unconditional "if" and "elseif" statements (PHP)	1007146
Avoid useless overriding Methods (PHP)	1007148
Avoid unassigned default values in Functions (PHP)	1007150
Avoid having variables without naming conventions (PHP)	1007212
Avoid having For-loops that use a function call in the test expression (PHP)	1007226

Avoid control structures without proper spacing before and after open\close braces - PSR2 (PHP)	1007228
Avoid Having control structures without proper switch case declarations (PSR2) (PHP)	1007230
Avoid having variables passed by reference when calling a function (PHP)	1007232
Avoid having inline control statements (PHP)	1007234
Avoid having Class Methods or Constructor without scope modifiers - Symfony STD (PHP)	1007236
Avoid having multiple classes defined in a single file - Symfony STD (PHP)	1007238
Avoid artifacts having object instantiation without parenthesis - Symfony STD (PHP)	1007240
CWE-311: Use sufficient SSL\TLS context (PHP)	1007248
Avoid files that declare both symbols and execute logic with side effects (PHP)	1007254

Rules using the PHP_CodeSniffer framework but implemented by CAST

Rule name	ID
Avoid using embedded CSS in Web Pages (PHP)	1007012
Avoid empty style definition (PHP)	1007034
Avoid artifacts with Object Instantiation in loops (PHP)	1007116
CWE-624: Avoid using eval expressions (PHP)	1007156
Avoid artifacts using exit and die expressions (PHP)	1007158
Avoid using variable without testing them for initialization (PHP)	1007160
Avoid having constructors with a return value (PHP)	1007172
Avoid using break or continue statements in loops with high cyclomatic complexity (PHP)	1007176
Avoid using size functions inside loops (PHP)	1007184
Avoid direct access to superglobals (PHP)	1007202
Avoid fetching database rows as array and accessing using subscript (PHP)	1007218
Avoid artifacts with Group By sql statement (PHP)	1007120
Avoid artifacts with "select *" Sql statement (PHP)	1007220
Avoid artifacts with sql statements referring more than 4 Tables (PHP)	1007118

phpcs-security-audit

This package integrates with the existing "Pear" code sniffer. This package is used to generate results for certain security related rules. More information about this package is available here: https://github.com/FloeDesignTechnologies/phpcs-security-audit. The licence agreement for this tool is available here: https://github.com/FloeDesignTechnologies/phpcs-security-audit/blob/master/LICENSE.

Rules calculated by the phpcs-security-audit tool

Rule name	ID
CWE-79: Avoid use of raw user input that can expose XSS vulnerability (PHP)	1007244
CWE-98: Avoid use of user input that can expose Stream Injection vulnerability (PHP)	1007246
CWE-624: Avoid preg_replace with /e option (PHP)	1007250
CWE-661: Avoid filesystem function calls without sanitizing user input (PHP)	1007252

PHPMD

More information about this tool is available here: http://phpmd.org/. The licence agreement for the PHPMD tool is detailed below:

PHP Depend

More information about this tool is available here: http://pdepend.org/. The licence agreement for the PHP Depend tool is available in the file "LICENSE.txt" delivered in the source folder of the tool and is detailed below: