The goal of this document is to provide, for each feature or major modification defined in a Requirement Specification document, the software design required by developers in order to implement the code, including:

- **Goals**
- **Strategy**
- **Design Architecture**
- **Design Details**
- **Future Development considerations**
- **Design Decisions**
- **Limitations**
- **Open Issues**
Table of Contents

1. Overview .................................................................................................................. 4
2. Strategy ...................................................................................................................... 4
  2.1 Guidelines ............................................................................................................. 4
3. Design ......................................................................................................................... 4
  3.1 Overview .............................................................................................................. 4
  3.2 Resources ............................................................................................................. 6
4. Detailed Design .......................................................................................................... 6
  4.1 Plug-in org.eclipse.php.core API ......................................................................... 6
    4.1.1 PHP Builder Extension .................................................................................. 6
    4.1.2 Include Path Variables .................................................................................. 9
    4.1.3 Workspace Model Listener .......................................................................... 11
  4.2 Plug-in org.eclipse.php.ui API ............................................................................ 13
    4.2.1 PHP Action Filter Contributor ..................................................................... 13
    4.2.2 PHP Folding Structure Provider ................................................................... 16
    4.2.3 PHP Content Assist Processor ..................................................................... 18
    4.2.4 PHP Editor Text Hover ............................................................................... 20
    4.2.5 PHP Element Filter ..................................................................................... 23
    4.2.6 PHP Hyperlink Detector ............................................................................... 25
    4.2.7 PHP Manual Director ................................................................................... 25
    4.2.8 PHP Manual URL ........................................................................................ 28
    4.2.9 PHP Outline Element Comparer ................................................................... 30
    4.2.10 PHP Preferences Page Block ...................................................................... 31
    ................................................................................................................................ 32
    4.2.11 PHP Tree Content Provider ........................................................................ 33
    4.2.12 PHP Wizard Page ........................................................................................ 35
  4.3 Plug-in org.eclipse.php.debug.core API ............................................................ 36
    4.3.1 PHP Debug Parameters Initializer .................................................................. 41
    4.3.2 PHP Debug Handler ....................................................................................... 43
    4.3.3 PHP Debug Message ..................................................................................... 43
    4.3.4 PHP exe .......................................................................................................... 45
  4.4 Plug-in org.eclipse.php.debug.deamon API ......................................................... 47
    4.4.1 Debug Communication Daemon .................................................................... 47
  4.5 Plug-in org.eclipse.php.debug.ui API ................................................................. 49
    4.5.1 PHP Debug Model Presentation ................................................................... 49
  4.6 Plug-in org.eclipse.php.server.core API ............................................................ 51
    4.6.1 httpServerLaunchDelegate .......................................................................... 51
  4.7 Plug-in org.eclipse.php.server.ui API .................................................................. 52
    4.7.1 serverTab ...................................................................................................... 52
    4.7.2 serverWizardFragment ................................................................................ 52
5. Testing ......................................................................................................................... 53
6. Future Development ................................................................................................... 53
7. Design Decisions ......................................................................................................... 53
  7.1 Removed the phpModel extension point ............................................................. 53
  7.2 BuildersInitializer: should the API use eclipse natures instead? ..................... 53
  7.3 Removed phpDebugActions extension point ....................................................... 54
  7.4 Removed phpEditorTextHoverDecorator extension point ................................ 54
  7.5 Extension point org.eclipse.php.ui.phplabeldecorator was removed .............. 54
  7.6 Extension name changes ....................................................................................... 54
8. Limitations ................................................................................................................... 55
9. Open Questions and Debates ...................................................................................... 55
1. Overview

This document describes the guidelines for selecting the PDT API as well as the extension points, interfaces and classes that form the PDT API. The document also contains a short description for each of the API entities.

2. Strategy

The PDT API consists of a set of Extension points including the interfaces and abstract base classes they define. All other classes and interfaces extended (or implemented) by Neon are not API and will be placed in package names that include internal in their name.

2.1 Guidelines

The PDT API uses the following guidelines:

- Minimize the plug-in exports (e.g. plug-in <export> tag). Otherwise, you risk a tighter coupling between plug-ins.
- Keep in mind that the <export> clause is enforced at runtime, but not in plug-in development environment. That means your code can compile correctly despite having references to a class in another plug-in that isn't visible because there is no <export> clause, resulting in a ClassNotFoundException at runtime.
- Eclipse implementation classes are in packages that include the internal in their name and export everything, allowing clients to reference its internal classes under the assumption that you accept the risk of these files being modified. The PDT API being part of eclipse tools uses that same guideline. If you only expect a single consumer of your extension point, it might be a candidate for a preferences or INI file configuration.

3. Design

3.1 Overview

The API consists of Extension points; therefore we start with its definition:

**Extension Point**

A basic rule for building modular software systems is to avoid tight coupling between components. If components are tightly integrated, it becomes difficult to assemble the pieces into different configurations or to replace a component with a different implementation without causing a ripple of changes across the system. Loose coupling in Eclipse is achieved partially through the mechanism of extensions and extension points. The simplest metaphor for describing extensions and extension points is electrical outlets. The outlet, or socket, is the extension point; the plug, or light bulb that connects to it, the extension. As with electric outlets, extension points come in a wide variety of shapes and sizes, and only the extensions that are designed for that particular extension point will fit.

---

1 See Java developers guide to eclipse, second edition page 252.
 When a plug-in wants to allow other plug-ins to extend or customize portions of its functionality, it will declare an extension point. The extension point declares a contract, typically a combination of XML markup and Java interfaces, that extensions must conform to. Plug-ins that want to connect to that extension point must implement that contract in their extension. The key attribute is that the plug-in being extended knows nothing about the plug-in that is connecting to it beyond the scope of that extension point contract. This allows plug-ins built by different individuals or companies to interact seamlessly, even without their knowing much about one another.

The Eclipse Platform has many applications of the extension and extension point concept. Some extensions are entirely declarative; that is, they contribute no code at all. For example, one extension point provides customized key bindings, and another defines custom file annotations, called markers; neither of these extension points requires any code on behalf of the extension.

Another category of extension points is for overriding the default behavior of a component. For example, the Java development tools include a code formatter but also supply an extension point for third-party code formatters to be plugged in. The resources plug-in has an extension point that allows certain plug-ins to replace the implementation of basic file operations, such as moving and deletion. Yet another category of extension points is used to group related elements in the user interface. For example, extension points for providing views, editors, and wizards to the UI allow the base UI plug-in to group common features, such as putting all import wizards into a single dialog, and to define a consistent way of presenting UI contributions from a wide variety of other plug-ins.
The PDT is a set of plug-ins each provides a set of functionality that can be extended or configured through its API. The plug-ins interact with each other by using the each other API. The following diagram describes the relations between the PDT plug-ins:

![Diagram showing PDT plug-ins dependencies]

**Figure 1 - PDT plugins dependencies**

### 3.2 Resources

The API does not use any external resources.

### 4. Detailed Design

The following subsections describe for each of the PDT plug-ins the extension points that form its API.

#### 4.1 Plug-in org.eclipse.php.core API

The org.eclipse.php.core plug-in is a module containing classes related to PHP model, document model, PHP parser, code formatter and PHP project.

##### 4.1.1 PHP Builder Extension

*Identifier: org.eclipse.php.core.phpBuilderExtension*
Since: 0.7

Description:

This extension point allows providing extensions to PHP builder. PHP builder is obliged to run all its extensions when it's being running on PHP project.

Configuration Markup:

```xml
<!ELEMENT extension (builder+)>  
<!ATTLIST extension
  point CDATA #REQUIRED
  id  CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT builder EMPTY>
<!ATTLIST builder
  id  CDATA #REQUIRED
  class CDATA #REQUIRED>

id - ID of the PHP builder extension.

class - Class of PHP builder extension which implements
  org.eclipse.php.core.project.build.IPHPBuilderExtension.

Declaration example:

```xml
<extension point="org.eclipse.php.core.phpBuilderExtensions">
  <builder
    class="org.eclipse.php.core.project.build.DefaultPHPBuilderExtension"
    id="org.eclipse.php.core.project.build.DefaultPHPBuilderExtension"/>
</extension>
```

API Information:

Each extension class must implement
org.eclipse.php.core.project.build.IPHPBuilderExtension.

Supplied Implementation:

Look at org.eclipse.php.core.project.build.DefaultPHPBuilderExtension
The following sequence diagram shows how this extension is used:

Created and initiated by Eclipse. This is the only builder defined for PHP project.

Original PHP builder delegates responsibility to its extensions.

Figure 2 - PHP Builder Extensions
4.1.2 Include Path Variables

Identifier: org.eclipse.php.core.includePathVariables

Since: 0.7

Description:

This extension point allows providing variables that will appear in an Include Path property page. Variable can be either set to the static value in the extension declaration, or initializer class can be provided in order to allow setting variable value at a runtime. Either static value or initializer class must be provided.

Configuration Markup:

```xml
<!ELEMENT extension (variable+)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT variable EMPTY>
<!ATTLIST variable
  name CDATA #REQUIRED
  value CDATA #IMPLIED
  initializer CDATA #IMPLIED>
```

- name - Variable name
- value - Variable value (if set statically through extension)
- initializer - Variable initializer class (if set dynamically)

Declaration example:

```xml
<extension point="org.eclipse.php.core.includePathVariables">
  <variable
    name="PEAR_LIBS"
    initializer="test.PearLibsVariableInitializer"/>

  <variable
    name="MY_HOME_LIBS"
    value="/home/michael/phplibs"/>
</extension>
```

test.PearLibsVariableInitializer must implement org.eclipse.php.core.IIncludePathVariableInitializer, and override abstract method initialize(String variable), where PEAR_LIBS must be resolved.

API Information:

Either static value or initializer class that implements org.eclipse.php.core.IIncludePathVariableInitializer must be provided.
Example of this extension in action:

These variables are contributed via includePathVariables extension point.
4.1.3 Workspace Model Listener

**Identifier:** org.eclipse.php.core.workspaceModelListener

**Since:** 0.7

**Description:**

This extension point allows providing listener which will receive event when PHP workspace model was changed.

See org.eclipse.php.core.phpModel.parser.PHPWorkspaceModelManager.

**Configuration Markup:**

```xml
<!ELEMENT extension (workspaceModelListener*)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id  CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT workspaceModelListener EMPTY>
<!ATTLIST workspaceModelListener
  id  CDATA #REQUIRED
  name CDATA #REQUIRED
  class CDATA #REQUIRED>
```

- **id** - Id of the workspace model listener
- **name** - Human readable name of the workspace model listener
- **class** - Workspace model listener class

**Declaration example:**

```xml
<extension point="org.eclipse.php.core.workspaceModelListener">
  <workspaceModelListener
    class="test.MyModelListener"
    id="test.MyModelListener"
    name="My Model Listener"/>
</extension>
```

**API Information:**

Each workspace model listener must implement org.eclipse.php.core.documentModel.IWorkspaceModelListener.
Here is very zoomed-in class diagram that may help to understand how to use this extension point:

```
PHPWorkspaceModelManager
- listeners : IWorkspaceModelListener
+ addWorkspaceModelListener(in listener : IWorkspaceModelListener)
+ removeWorkspaceModelListener(in listener : IWorkspaceModelListener)
+ getModelForProject(in project, in forceCreate : boolean)

listeners : IWorkspaceModelListener

+ interface
  IWorkspaceModelListener
+ projectModelAdded(in project)
+ projectModelRemoved(in project)
+ projectModelChanged(in project)

ProjectOutlineContentProvider

This class listens to project model changes in order to invoke refresh on Project Outline View.
```
4.2 Plug-in org.eclipse.php.ui API

4.2.1 PHP Action Filter Contributor

**Identifier:** org.eclipse.php.ui.actionFilterContributors

**Since:** 0.7

**Description:**

This extension point allows providing contributors to IActionFilter in order to perform more complicated tests on whether the action should be visible/enabled or not. You must add parameter under the visibility/enablement element of relevant action, which is called "actionFilterContributorId", and has a value that equals to id of this extension.

**Configuration Markup:**

```
<!ELEMENT extension (contributor+)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT contributor EMPTY>
<!ATTLIST contributor
  id CDATA #REQUIRED
  name CDATA #IMPLIED
  class CDATA #REQUIRED>
```

- **id** - ID of this PHP action filter contributor
- **name** - Human readable name of this PHP action filter contributor
- **class** - Class of this PHP action filter contributor

**Declaration example:**

Suppose declaration of the action is:

```
<extension point="org.eclipse.ui.popupMenus">
  <objectContribution id="test.MyObjectContribution"
    objectClass="org.eclipse.core.resources.IFile">
    <visibility>
      <objectState name="actionFilterContributorId"
        value="test.MyActionFilterContributor"/>
    </visibility>
    <action class="test.MyAction"
      id="test.MyAction"
      label="My Action"/>
  </objectContribution>
</extension>
```
Then the action filter contributor must be defined as follows:

```xml
<extension point="org.eclipse.php.ui.actionFilterContributors">
  <contributor id="test.MyActionFilterContributor"
    class="test.MyActionFilterContributor"
    name="My Action Filter Contributor"/>
</extension>
```

This guarantees, that when testing the visibility of My Action, testAttribute method of test.MyActionFilterContributor will be called for performing custom testing.

**API Information:**

Extensions must implement org.eclipse.php.ui.generic.actionFilter.IActionFilterContributor interface.

**Supplied Implementation:**

Look at: AddPHPDocActionFilterContributor and CodeDataActionFilterContributor.

Example of this extension in action:
The following sequence diagram represents stages of Action Filter Contributor extension activation:

GeneralActionFilter is activated by the popup menu manager. When user right-clicks on some element in the view, menu manager tries to transform selected element to IActionFilter class using adapter (in our case PHPCodeData is transformed into GeneralActionFilter). Menu manager, then, calls testAttribute() method of the action filter in order to determine whether the element should be visible or enabled.

GeneralActionFilter checks whether the test attribute name equals to "actionFilterContributorId". If yes, it assumes the value of the test attribute is the action filter contributor's ID. Then it looks for this action filter contributor in the ActionFilterContributorRegistry, and delegates to it.

**Figure 3 - Action Filter Contributor extension activation**
### 4.2.2 PHP Folding Structure Provider

**Identifier:** org.eclipse.php.ui.foldingStructureProviders

**Since:** 0.7

**Description:**

Contributions to this extension point define folding structures for the PHP editor. That is, they define the regions of a PHP source file that can be folded away. See org.eclipse.jface.text.source.ProjectionViewer for reference.

Extensions may optionally contribute a preference block which will appear on the PHP editor preference page.

**Configuration Markup:**

```xml
<!ELEMENT extension (provider)>
<!ATTLIST extension
point CDATA #REQUIRED
id CDATA #IMPLIED
name CDATA #IMPLIED>

<!ELEMENT provider EMPTY>
<!ATTLIST provider
id CDATA #REQUIRED
name CDATA #IMPLIED
class CDATA #REQUIRED
preferencesClass CDATA #IMPLIED>

id - The unique identifier of this provider.
name - The name of this provider. If none is given, the id is used instead.
class - An implementation of IStructuredTextFoldingProvider
preferencesClass - An implementation of org.eclipse.php.ui.folding.IPHPFoldingPreferenceBlock
```

**Declaration example:**

See org.eclipse.php.internal.ui.folding.DefaultPHPFoldingStructureProvider for an example.

**Supplied Implementation:**

- org.eclipse.php.internal.ui.folding.DefaultPHPFoldingStructureProvider - provides the default folding structure for the PHP editor.
- org.eclipse.php.internal.ui.folding.DefaultPHPFoldingPreferenceBlock - provides the preference block for the default structure provider.

**Example of this extension in action:**
Provided folding structures appear in Preference Page as checkboxes.

- Enable folding
- Usually fold these elements:
  - Classes
  - Functions
  - PHPDoc
4.2.3 PHP Content Assist Processor

Identifier: org.eclipse.php.ui.phpContentAssistProcessor

Since: 0.7

Description:

This extension point allows providing additional content assist processors for PHP editor. For more information of what is content assist processor, refer to org.eclipse.jface.text.contentassist.IContentAssistProcessor interface.

Configuration Markup:

```xml
<!ELEMENT extension (processor*)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT processor EMPTY>
<!ATTLIST processor
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  class CDATA #REQUIRED>

  id - ID of this PHP content assist processor
  name - Human readable of this PHP content assist processor
  class - Class of this PHP content assist processor

Declaration example:

```xml
<extension point="org.eclipse.php.ui.phpContentAssistProcessor">
  <processor class="test.MyContentAssistProcessor"
    id="test.MyContentAssistProcessor"
    name="My PHP Content Assist Processor"/>
</extension>

API Information:

PHP content assist processor class must implement
org.eclipse.php.ui.editor.contentassist.IContentAssistProccesorForPHP
Example of this extension in action:

```php
$client = new SoapClient(
   "\gibraltar\michael\GoogleSearch.wSDL"
);
$client->
```

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoapClient</td>
<td>Calls a SOAP function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_name string arguments array[optional] options array[optional] input_headers array[optional]</td>
</tr>
</tbody>
</table>

Completion suggestions for WSDL are provided via phpContentAssistProcessor extension.
4.2.4 PHP Editor Text Hover

**Identifier:** org.eclipse.php.ui.phpEditorTextHovers

**Since:** 0.7

**Description:**

This extension point allows adding new PHP Text Hovers to the editor. For more information of what is Text Hover, refer to org.eclipse.jface.text.ITextHover interface. The most suitable for current context hover will be chosen by org.eclipse.php.ui.editor.hover.BestMatchHover at runtime.

**Configuration Markup:**

```xml
<!ELEMENT extension (hover*)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT hover EMPTY>
<!ATTLIST hover
  id CDATA #REQUIRED
  label CDATA #IMPLIED
  description CDATA #IMPLIED
  class CDATA #REQUIRED
  activate (true | false) "false"
  priority CDATA #IMPLIED>
```

- **id** - ID of the PHP Editor Text Hover.
- **label** - Label of the PHP Editor Text Hover.
- **description** - Description of the PHP Editor Text Hover.
- **class** - Class of the PHP Editor Text Hover.
- **activate** - if the attribute is set to "true" it will force this plug-in to be loaded on hover activation.
- **priority** - Priority allows fixing the order of how text hovers are processed by PHP editor. The more is this number, the more important is the text hover. Known priorities are:
  - 1000 – Best Match Hover
  - 500 – Problem Hover
  - 400 – PHP Annotation Hover
  - 300 – PHP Source Hover
  - 200 – PHP Debug Text Hover

First hover which returns non-null value from getHoverInfo method is actually active.

**Declaration example:**

```xml
<extension point="org.eclipse.php.ui.phpEditorTextHovers">
  <hover class="org.eclipse.php.ui.editor.hover.PHPDebugTextHover">
```
id="org.eclipse.php.ui.editor.hover.PHPDebugTextHover"
name="PHP Debug Text Hover"
priority="1"/
</extension>
API Information:

Plug-in that want to extend this extension point must implement this interface:
org.eclipse.php.ui.editor.hover.IPHPTextHover.

Supplied Implementation:

org.eclipse.php.ui.editor.hover.PHPAnnotationTextHover - used for displaying information on PHP element.
org.eclipse.php.ui.editor.hover.PHPSourceTextHover - used for displaying source code of PHP element declaration (active, when CTRL button is pressed).
org.eclipse.php.ui.editor.hover.BestMatchHover, which is a composite of all PHP Text Hovers. It invokes the most suitable for current context text hover each time.

Example of this extension in action:

This text hover (PHPSourceTextHover) shows source code of the element that is currently under the mouse (activation key is CTRL).
4.2.5 PHP Element Filter

Identifier: org.eclipse.php.ui.phpElementFilters

Since: 0.7

Description:

This extension point allows adding new types of PHP view filters. Contributed types of PHP view filters are collected through `FilterDescriptor.getFilterDescriptors(String viewId)`. PHP Explorer View provides an ability to select active filters using special dialog, which can be invoked by the menu action "Filters..." (implemented in `CustomFiltersDialog`).

Configuration Markup:

```xml
<!ELEMENT extension (filter*)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT filter EMPTY>
<!ATTLIST filter
  id CDATA #IMPLIED
  name CDATA #IMPLIED
  description CDATA #IMPLIED
  targetId CDATA #IMPLIED
  enabled CDATA #IMPLIED
  pattern CDATA #IMPLIED
  class CDATA #IMPLIED>
```

- **id** - ID of the PHP view filter.
- **name** - Human readable name of the PHP view filter.
- **description** - Human readable description of the PHP view filter.
- **targetId** - ID of a view that this PHP view filter will apply to.
- **enabled** - Whether this filter is enabled by default.
- **pattern** - File name pattern that this filter represents.
- **class** - A class that implements this PHP view filter.
Declaration example:

```xml
<extension point="org.eclipse.php.ui.phpElementFilters">
  <filter id="org.eclipse.php.ui.explorer.dotFilesFilter"
    targetId="org.eclipse.php.ui.explorer"
    name="Hide \\.*\ Resources"
    enabled="true"
    description="Hides resources whose file name starts with a dot (.*))"
    pattern="\.*"/>
</extension>
```

API Information:

PHP view filter can be defined either using class, which extends org.eclipse.jface.viewers.ViewerFilter or using a pattern string (the filter will try to perform a file name match, using the pattern specified in the extension point declaration; see NamePatternFilter class).

Supplied Implementation:

Many basic filters are contributed in PHP UI Plug-in; for the full list refer to the plug-in's manifest file.

Example of this extension in action:

![PHP Element Filters dialog](image)

These filters are added using extension point phpElementFilters. This dialog is a part of Explorer view (to open: Menu -> Filters...
4.2.6 PHP Hyperlink Detector

Identifier: org.eclipse.php.ui.phpHyperlinkDetector

Since: 0.7

Description:

This extension provides support for detecting and displaying hyperlinks in a PHP text viewer. Contributed hyperlink detectors try to find a hyperlink at a given location in a PHP text viewer, the first of them which succeeds to resolve current element becomes active at that point.

Configuration Markup:

```xml
<!ELEMENT extension (detector+)>
<!ATTLIST extension
   point CDATA #REQUIRED
   id CDATA #IMPLIED
   name CDATA #IMPLIED>

<!ELEMENT detector EMPTY>
<!ATTLIST detector
   id CDATA #REQUIRED
   name CDATA #REQUIRED
   class CDATA #REQUIRED>
```

- **id** - ID of this PHP hyperlink detector.
- **name** - Name of this PHP hyperlink detector.
- **class** - Class of this PHP hyperlink detector.

Declaration example:

```xml
<extension point="org.eclipse.php.ui.phpHyperlinkDetector">
   <detector
      class="com.foo.bar.MyCodeHyperlinkDetector"
      id="com.foo.bar.MyCodeHyperlinkDetector"
      name="My Code Hyperlink Detector"/>
</extension>
```

API Information:

Plug-in that wishes to extend this extension point must implement interface: org.eclipse.php.ui.editor.hover.IHyperlinkDetectorForPHP

Supplied Implementation:

Default PHP text viewer hyperlink detector (PHPCodeHyperlinkDetector) is contributed directly in the PHP structured text viewer configuration class.

4.2.7 PHP Manual Director

R&D Department

Since: 0.7

Description:

PHP Manual director defines mapping between PHP element name and site path. Most probably, a site containing PHP documentation will have a standard file hierarchy generated by PHPDoc, therefore providing new PHP manual site does not necessarily require defining new PHP manual director (a default org.eclipse.php.ui.phpManualDirector can be used).

Configuration Markup:

```xml
<!ELEMENT extension (director*)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT director (element*)>
<!ATTLIST director
  id CDATA #REQUIRED
  label CDATA #REQUIRED>

  id  - ID of this PHP manual director.
  label  - Human readable label of this PHP manual director.

<!ELEMENT element EMPTY>
<!ATTLIST element
  name CDATA #REQUIRED
  path CDATA #REQUIRED>

  name  - The name of the PHP element.
  path  - Path on the site that used for reference to PHP element manual.

Declaration example:

```xml
<extension point="org.eclipse.php.ui.phpManualDirectors">
  <director id="org.eclipse.php.ui.phpManualDirector"
    label="Default PHP Manual Director">
    <element name="if"
      path="language.control-structures#control-structures.if"/>
    <element name="*"
      path="function.%NAME.%EXT"/>
  </director>
</extension>
```
API Information:

The following variable placeholders can be used in the element attributes:

- `%NAME` - PHP element name (use '*' for all elements)
- `%EXT` - Site file suffix (defined in `phpManualSites` extension point)

Supplied Implementation:

4.2.8 PHP Manual URL

**Identifier:** org.eclipse.php.ui.phpManualSites

**Since:** 0.7

**Description:**

This extension point allows introducing the site that can be used for accessing PHP manual. A list of PHP manual sites can be seen in the PHP Manual preference page; only one of them is active at a time. Selected PHP manual site will be used when user presses F3 when PHP element is selected, or when he right-clicks it, and selects "Open PHP Manual page".

**Configuration Markup:**

```xml
<!ELEMENT extension (site*)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT site EMPTY>
<!ATTLIST site
  url CDATA #REQUIRED
  director CDATA #IMPLIED
  extension CDATA "php"
  label CDATA #REQUIRED
  id CDATA #REQUIRED>
```

- **director** - ID of the PHP Manual Director (see phpManualDirectors extension point description).
- **extension** - File extensions on the site (for example: .php, .html)
- **label** - Human readable label of this PHP manual site.
- **id** - ID of this PHP manual site.

**Declaration example:**

```xml
<extension point="org.eclipse.php.ui.phpManualSites">
    director="org.eclipse.php.ui.phpManualDirector"/>
</extension>
```

**API Information:**

No API information, this extension point provides only configuration entries.

**Supplied Implementation:**


**Example of this extension point in action:**

List of provided Manual sites can be seen in preference page.
This opens PHP manual on the selected element, using currently configured manual site.
4.2.9 PHP Outline Element Comparer

**Identifier:** org.eclipse.php.ui.phpOutlineElementComparers

**Since:** 0.7

**Description:**

This extension point allows defining comparer (yes, it's called comparer, see org.eclipse.jface.viewers.IElementComparer) for elements displayed in PHP outline. This extension point interface is used to compare elements in a viewer for equality, and to provide the hash code for an element. This allows the client of the viewer to specify different equality criteria and a different hash code implementation than the `equals` and `hashCode` implementations of the elements themselves.

**Configuration Markup:**

```xml
<!ELEMENT extension (comparer+)>
<!ATTLIST extension
    point CDATA #REQUIRED
    id CDATA #IMPLIED
    name CDATA #IMPLIED>

<!ELEMENT comparer EMPTY>
<!ATTLIST comparer
    name CDATA #REQUIRED
    id CDATA #REQUIRED
    class CDATA #REQUIRED>
```

- **name** - Name of this element comparer.
- **id** - Name of this element comparer.
- **class** - Name of this element comparer, that implements org.eclipse.php.ui.util.IPHPOutlineElementComparer

**Declaration example:**

```xml
<extension point="org.eclipse.php.ui.phpOutlineElementComparers">
    <comparer class="test.MyPHPElementComparer"
        id="test.MyPHPElementComparer"
        name="My PHP Element Comparer"/>
</extension>
```

**API Information:**

PHP outline element comparers must implement interface org.eclipse.php.ui.util.IPHPOutlineElementComparer.
4.2.10 PHP Preferences Page Block

Identifier: org.eclipse.php.ui.phpPreferencePageBlocks

Since: 0.7

Description:

This extensions point allows contributing parts of PHP preference page. Each part is a composite that is added to the original page. In order to use this extension, you must make sure that the preference page you are contributing this extension to extends AbstractPHPPropertyPreferencePage.

Configuration Markup:

```xml
<!ELEMENT extension (block+)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT block EMPTY>
<!ATTLIST block
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  pageId CDATA #REQUIRED
  class CDATA #REQUIRED>
```

- **id** - The unique id for this extension
- **name** - The name of this extension. Note that the name will also effect the location of the add-on. The add-ons will be added according to their names in a lexicographic order.
- **pageId** - The id of the PHP preferences/property page that will handle this extension when the page is created.
- **class** - The class that will generate the Composite add-on for the preferences page.

Declaration example:

```xml
<extension point="org.eclipse.php.ui.phpPreferencePageBlocks">
  <block class="test.MyPHPDebugPreferencePageAddon"
    id="test.MyPHPDebugPreferencePageAddon"
    name="PHP Debug Preference Page Add-on"
    pageId="org.eclipse.php.debug.ui.preferencesPhpDebugPreferencePage"
  />
</extension>
```
Extensions must implement `org.eclipse.php.ui.preferences.IPHPPreferencePageBlock`. Abstract class `AbstractPHPPreferencePageBlock` can be used for convenience.

Example of this extension in action:

This preference page block is contributed through `phpPreferencesPageBlocks` extension point. Each block is actually a SWT Composite, so it's easily inserted into the original page.
4.2.11 PHP Tree Content Provider

**Identifier**: org.eclipse.php.ui.phpTreeContentProviders

**Since**: 0.7

**Description**:
This extension points allows providing content providers for a Tree Viewer used for displaying PHP elements (PHP Explorer, Outline, Project Outline, etc'). Standard PHP Element Content Provider (the default one) collects all content providers contributed via extension point, and concatenates their contents.

**Configuration Markup**:

```xml
<!ELEMENT extension (provider)+>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT provider EMPTY>
<!ATTLIST provider
  class CDATA #REQUIRED
  targetId CDATA #IMPLIED>
```

- **class** - PHP tree content provider's class
- **targetId** - A comma-separated list of tree viewers, this content provider is applicable to.

**Declaration example**:

```xml
<extension point="org.eclipse.php.ui.phpTreeContentProviders">
  <provider
class="org.eclipse.php.ui.treecontent.IncludePathTreeContent"
targetId="org.eclipse.php.ui.explorer"/>
</extension>
```

**API Information**:

PHP tree content provider class must implement org.eclipse.php.ui.treecontent.IPHPTreeContentProvider.

**Supplied Implementation**:

Look at org.eclipse.php.ui.treecontent.IncludePathTreeContent, which is contributed to PHP Explorer.

**Example of this extension in action**:

WSDL entries in Outline View are provided through PHP Tree Content Provider extension point.
4.2.12 PHP Wizard Page

Identifier: org.eclipse.php.ui.phpWizardPages

Since: 0.7

Description:

This extension point is used for providing additional pages to PHP wizards.

Configuration Markup:

```xml
<!ELEMENT extension (page)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT page EMPTY>
<!ATTLIST page
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  targetId CDATA #REQUIRED
  class CDATA #REQUIRED>
```

- **id** - Id of this PHP wizard page.
- **name** - Human readable name of this PHP Wizard page.
- **targetId** - Wizard ID to whom this page is contributed.
- **class** - Class of this PHP Wizard Page (must extend org.eclipse.jface.wizard.IWizardPage).

Declaration example:

The following example will add "My Wizard Page" to the PHP project creation wizard:

```xml
<extension point="org.eclipse.php.ui.phpWizardPages">
  <page class="test.MyWizardPage"
    id="test.MyWizardPage"
    name="My Wizard Page"
    targetId="org.eclipse.php.project.ui.wizards.PHPProjectCreationWizard"/>
</extension>
```

API Information:

PHP Wizard page must extend class org.eclipse.jface.wizard.IWizardPage. Wizard must use PHPWizardPagesRegistry in order to retrieve all contributed pages.
4.3 Plug-in org.eclipse.php.debug.core API

This plug-in implements the communication of the PDT with the debugger executable and manages debug sessions. The following diagrams show the sequence of debug events and the extension points for enhancing the debug functionality:

Figure 4 - Launch debug session using internal debugger
API Extension Point: phpDebugParametersInitializer:
This extension point is used to create the appropriate instance of an IDebugParametersInitializer.
(1) In PDT the DefaultDebugParametersInitializer is used and allows SSL.
(2) In Neon the AdvancedDebugParametersInitializer is used and allows SSL.
(3) When launching a profile session, we use the ProfilerParametersInitializer.

Figure 5 - Launch debug session using remote debug server
The following diagram shows the process of the debugger initializing the communication with the PDT and how the PDT uses the phpDebugMessages extension point to get the appropriate IDebugMessage implementation for deserializing the debug message received from the debugger.

**Figure 6 - process asynchronous request for starting new debug session.**
The following sequence diagram shows the process of handling a DebugSessionStartedNotification message and how the phpDebugHandlers extension point creates a new IDebugHandler instance to process the IDebugMessage received in the previous diagram:

Remote debugger is actually a manager of debug session. When it's requested to handle a notification message, it calls relevant method of IDebugHandler which is attached to it.

In this scenario the message is of type: DebugSessionStartedNotification

InputMessageHandler runs while debug session is alive. First it receives the new arrived message from queueOut, then it processes the message according to type.

The PHPDebugTarget creates a remote debugger and set it to the debugConnThread (See "PHPDebugTarget" diagram for details).

Figure 7 - processing DebugSessionStartedNotification message

Note: a similar sequence also represents the process handling for other debug messages.

The following sequence diagram shows the creating DebugHandler by the phpDebugTarget:
The same DebugParametersInitializer, that for used for initiation of debug session is returned.

Figure 8 - creating DebugHandler
The following subsections elaborate on each of the extension point:

### 4.3.1 PHP Debug Parameters Initializer

**Identifier**: org.eclipse.php.debug.core.phpDebugParametersInitializer

**Since**: 0.7

**Description**:

In order to initiate debug session, special parameters are being sent to the PHP Debugger, either via URL or via environment variables (for debugging through PHP CGI binary). Debug session configuration can be changed also using these parameters. PHP Debug parameters initializer provides a method for generating request to PHP Debugger. If debug session must be different from the standard one (profile session, for example), most probably different parameters initializer must be used. In addition, PHP Debug Handler can be attached to the initializer in order to maintain mapping between debug session initialization and handling. Correct PHP Debug parameters initializer is chosen by its mode and launch configuration type ID.

**Configuration Markup**:

```xml
<!ELEMENT extension (initializer*)>
<!ATTLIST extension
    point CDATA #REQUIRED
    id CDATA #IMPLIED
    name CDATA #IMPLIED>

<!ELEMENT initializer EMPTY>
<!ATTLIST initializer
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    class CDATA #REQUIRED
    handler CDATA #IMPLIED
    mode (run|debug|profile) "debug"
    launchConfigurationType CDATA #IMPLIED>
```

- **id** - The debug parameters initializer ID
- **name** - Human readable name of the debug parameters initializer.
- **class** - Class of this debug parameters initializer, which implements org.eclipse.php.debug.core.debugger.parameters.IDebugParametersInitializer.
- **handler** - ID of the debug handler associated with this parameters initializer (see extension point: org.eclipse.php.debug.core.phpDebugHandlers)
- **mode** - Debug mode this parameters initializer should work with.
- **launchConfigurationType** - Launch configuration type ID this parameters initializer should work with (see extension point: org.eclipse.debug.core.launchConfigurationTypes).
Declaration example:

```xml
<extension point="org.eclipse.php.debug.core.phpDebugParametersInitializers">
  <initializer
class="org.eclipse.php.debug.core.debugger.parameters.DefaultDebugParametersInitializer"
    id="org.eclipse.php.debug.core.defaultInitializer"
    name="Default Debug Parameters Initializer"/>
</extension>
```

API Information:

Each PHP Debug parameters initializer must implement interface org.eclipse.php.debug.core.debugger.parameters.IDebugParametersInitializer.

Supplied Implementation:


Figure 3 and 7 (above) shows the usage of phpDebugParameterInitializer.
### 4.3.2 PHP Debug Handler

**Identifier:** org.eclipse.php.debug.core.phpDebugHandlers

**Since:** 0.7

**Description:**
This extension point is used for contributing PHP debug handlers. Each debug handler provides methods for handling various debug messages received from PHP Debug Server. See org.eclipse.php.debug.core.debugger.IDebugHandler.

**Configuration Markup:**

```xml
<!ELEMENT extension (handler*)>
<!ATTLIST extension
point CDATA #REQUIRED
id CDATA #IMPLIED
name CDATA #IMPLIED>

<!ELEMENT handler EMPTY>
<!ATTLIST handler
id CDATA #REQUIRED
name CDATA #REQUIRED
class CDATA #REQUIRED>
```

- **id** - ID of the debug handler.
- **name** - Human readable name of the debug handler.
- **class** - Debug handler class, which implements org.eclipse.php.debug.core.debugger.IDebugHandler.

**Declaration example:**

This debug handler adds functionality of PHP profiler to the basic PHP debug handler:

```xml
<extension point="org.eclipse.php.debug.core.phpDebugHandlers">
  <message class="test.PHPProfiler"
    id="test.PHPProfiler"
    name="PHP Profiler"/>
</extension>
```

**API Information:**

Plug-ins that want to extend this extension point must implement this interface:

org.eclipse.php.debug.core.debugger.IDebugHandler.

### 4.3.3 PHP Debug Message

**Identifier:** org.eclipse.php.debug.core.phpDebugMessages
Since: 0.7

Description:

This extension point is used for contributing PHP Debugger message types. Each message has two basic methods for serialization/de-serialization, allowing to transfer structured information between PDT and PHP Debug Server. There is a predefined set of Debug Messages, which are supported by current PHP Debugger protocol; providing new message will require changes in the protocol.

Configuration Markup:

```xml
<!ELEMENT extension (message*)>
<!ATTLIST extension
    point CDATA #REQUIRED
    id CDATA #IMPLIED
    name CDATA #IMPLIED>

<!ELEMENT message EMPTY>
<!ATTLIST message
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    class CDATA #REQUIRED
    handler CDATA #IMPLIED>
```

- **id** - ID of this message.
- **name** - Name of this message.
- **class** - Class of this message that implements `IDebugMessage`.
- **handler** - Handler class for this message that implements `IDebugMessageHandler`.

Declaration example:

```xml
<extension point="org.eclipse.php.debug.core.phpDebugMessages">
    <message
        class="org.eclipse.php.debug.core.debugger.messages.AddBreakpointRequest"
        id="org.eclipse.php.debug.core.debugger.messages.AddBreakpointRequest"
        name="Add Breakpoint Request"/>
</extension>
```

API Information:

Plug-ins that want to extend this extension point must implement this interface:

`org.eclipse.php.debug.core.debugger.messages.IDebugMessage`.
4.3.4 PHP exe

Identifier: org.eclipse.php.debug.core.phpExe

Since: 0.7

Description:

This extension point allows providing PHP binaries for internal debugging. PHP executable must be configured to work with Zend Debugger, i.e. file "php.ini" must be placed in the same directory where PHP binary resides, and must contain at least the following entry:

zend_extension=<path to the Zend Debugger extension>

Configuration Markup:

```xml
<!ELEMENT extension (phpExe)>  
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT phpExe EMPTY>
<!ATTLIST phpExe
  name CDATA #IMPLIED
  location CDATA #REQUIRED
  version CDATA #IMPLIED
  default (true | false) >

  name - Human readable name of the PHP executable (for example: "PHP 5"). This name will appear in the preferences page.
  location - Path to the PHP executable. The binary itself may be contributed either through the plug-in or through its fragment (which is recommended).
  version - Version of PHP (<major>.<minor>.<bug-fix>)
  default - Whether this PHP binary should be enabled by default.

Declaration example:

```xml
<extension point="org.eclipse.php.debug.core.phpExe">
  <phpExe location="/resources/php4/php"
    name="PHP4"
    version="4.4.4"/>

  <phpExe
    default="true"
    location="/resources/php5/php"
    name="PHP5"
    version="5.2.0"/>
</extension>
```

Example of this extension in action:
Figure 9 - selecting a PHP executable for debug session
4.4 Plug-in org.eclipse.php.debug.deamon API

This plug-in implements the communication of the PDT with the debugger executable and manages debug sessions.

4.4.1 Debug Communication Daemon

**Identifier:** org.eclipse.php.debug.daemon.debugCommunicationDaemon

**Since:** 0.7

**Description:**
Debug communication daemon supplies an extension point for any daemon class that is needed for starting or maintaining debug sessions with a PHP debugger. Debug connection daemon listens to incoming connection from PHP debug server, which is a request for starting new debug session.

**Configuration Markup:**

```xml
<!ELEMENT extension (daemon)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT daemon EMPTY>
<!ATTLIST daemon
  id CDATA #REQUIRED
  class CDATA #REQUIRED
  name CDATA #IMPLIED>
```

**Communication Daemon**

- **id** - A unique identifier for this daemon extension.
- **class** - An ICommunicationDaemon implementation.
- **name** - The name of this daemon [optional].

**Declaration example:**

```xml
<extension
  point="org.eclipse.php.debug.daemon.debugCommunicationDaemon">
  <daemon class="test.MyDebuggerCommunicationDaemon"
         id="test.MyDebuggerCommunicationDaemon"
         name="Advanced Debug Daemon"/>
</extension>
```

**API Information:**
Debug Communication Daemon class must implement
org.eclipse.php.debug.daemon.communication.ICommunicationDaemon.

**Supplied Implementation:**

org.eclipse.php.debug.core.communication.DebuggerCommunicationDaemon
4.5 Plug-in org.eclipse.php.debug.ui API

4.5.1 PHP Debug Model Presentation

Identifier: org.eclipse.php.debug.ui.phpDebugModelPresentations

Since: 0.7

Description:

This extension point allows adding new PHP debug model presentations, or extending the default one; only one presentation will be active. The PHPModelPresentationRegistry class decides which of the registered PHP debug model presentations will be used, giving priority to extensions defined in org.eclipse.php.debug.ui plug-in. A debug model presentation is responsible for providing labels, images, and editors for elements in a specific debug model (see org.eclipse.debug.ui.debugModelPresentations extension point).

Configuration Markup:

```xml
<!ELEMENT extension (phpDebugModelPresentation)>  
<!ATTLIST extension
 point CDATA #REQUIRED
 id   CDATA #IMPLIED
 name CDATA #IMPLIED>

<!ELEMENT phpDebugModelPresentation EMPTY>
<!ATTLIST phpDebugModelPresentation
 id   CDATA #REQUIRED
 class CDATA #REQUIRED
 detailsViewerConfiguration CDATA #IMPLIED>
```

- **id** - the identifier of the debug model this presentation is responsible for.

- **class** - fully qualified name of a Java class that implements the org.eclipse.debug.ui.IDebugModelPresentation interface. Since 3.1, debug model presentations may optionally implement IColorProvider and IFontProvider to override default fonts and colors for debug elements.

- **detailsViewerConfiguration** - the fully qualified name of the Java class that is an instance of org.eclipse.jface.text.source.SourceViewerConfiguration. When specified, the source viewer configuration will be used in the "details" area of the variables and expressions view when displaying the details of an element from the debug model associated with this debug model presentation. When unspecified, a default configuration is used.
Declaration example:

```xml
<extension
    point="org.eclipse.php.debug.ui.phpDebugModelPresentations">
    <phpDebugModelPresentation
        class="org.eclipse.php.debug.ui.presentation.PHPModelPresentation"
        id="org.eclipse.php.debug.ui.presentation.phpModelPresentation"/>
</extension>
```

API Information:

**PHP Debug Model Presentation class** must implement
`org.eclipse.debug.ui.IDebugModelPresentation`.

Supplied Implementation:

`org.eclipse.php.debug.ui.presentation.PHPModelPresentation`

Example of this extension in action:

Here, PHP Debug Model Presentation extension is used for changing decoration in this view.
4.6 Plug-in org.eclipse.php.server.core API

4.6.1 httpServerLaunchDelegate

Type:

Description:

Example:

How to use:
4.7 Plug-in org.eclipse.php.server.ui API

4.7.1 serverTab

Type:

Description:

Example:

How to use:

4.7.2 serverWizardFragment

Type:

Description:

Example:

How to use:
5. **Testing**

This section should be used to define the functionality to be tested at the Unit Test level.

<table>
<thead>
<tr>
<th>Test Item</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. **Future Development**

Any requirement for future development of this feature should be described in this section.

7. **Design Decisions**

7.1 **Removed the phpModel extension point**

This extension point was redundant, since all PHP elements are really provided through content assist, hyperlink detector and PHP tree content provider extension points.

**7.2 BuildersInitializer: should the API use eclipse natures instead?**

The extension point "buildersInitializers" collects all builders (CodeAnalyzer, WSDL, JavaBridge, etc) at the project creation time. This produces a list of all builders related to the PHP project nature and writes them into ".project" file. Thus, Eclipse "knows" what builders it must run after each update of any resource in this project.

This is not a standard Eclipse way. The correct way is to provide a new project nature, say CodeAnalyzer project nature, which has a relation of "requires-nature" to the original PHP project nature ("the main purpose of project natures is to create an association between a project and a given tool, plug-in, or feature set". See: [http://www.eclipse.org/articles/Article-Builders/builders.html](http://www.eclipse.org/articles/Article-Builders/builders.html)). Then, in the nature configure() and deconfigure() class methods add/remove relevant builder to/from the list of existing project builders. Configure and deconfigure methods run when the nature ID is added/removed to/from the ".project" file **programmatically** via calling to project description update methods.

**Problem 1:** if we create a new nature, it won't be added to the project automatically. Eclipse requires a new property page (UI) that allows user to enable the additional nature on selected project. See, for example, ways how other features do it:

1) Adding the Derby nature to a Java project ([http://db.apache.org/derby/integrate/plugin_help/nature.html](http://db.apache.org/derby/integrate/plugin_help/nature.html))
2) Add/Remove Tapestry Project Nature

3) Toggle ANTLR plug-in nature
(http://antreclipse.sourceforge.net/images/toggle-nature.png)

We have two options here:

1) Eclipse way: remove "buildersInitializers" extension point, connect "Code Analyzer", "Java Bridge" and "WSDL" builders via their project natures. Add property page (or pages) allowing to enable these natures on selected projects. I must mention, that when creating a new PHP project, additional natures will not be added automatically, so we must create some extension point for the "New PHP Project Wizard", which will allow to configure additional project natures (may be add the same functionality to the preference page as well). Other crazy solution may be to create a new "New Neon Project Wizard", which extends basic PHP Wizard and adds new functionality through it (the way as PDE new plug-in wizard extends new Java project wizard, and adds PDE project nature along with Java nature there).

2) Stay with "buildersInitializers"

Final decision is: phpBuilderExtensions extension point.

**7.3 Removed phpDebugActions extension point**

Reason:
- Extension point goal wasn’t clear.
- There where not any usages of this extension point.

**7.4 Removed phpEditorTextHoverDecorator extension point.**

Reason:
- There was a suspect that this extension is useless.
- Replaced by extending phpEditorTextHovers extension point (adding priorities, ability to override).

**7.5 Extension point org.eclipse.php.ui.phplabeldecorator was removed.**

Reason: replaced by standard extension point: org.eclipse.ui.decorators.

**7.6 Extension name changes**

Some extension points where renamed either according to the naming convention, or according to their usage.

- phpDebugPreferencesAddon was renamed to phpPreferencePageBlocks, and moved to PHP UI plug-in (generalized).
- actionFilterDelegatorForPHPCodeData was renamed to actionFilterContributors (strange name).
- includePathVariableInitializer was renamed to includePathVariables (it provides variables, not initializers).
contentAssistProcessorForPHP was renamed to phpContentAssistProcessor.
hyperlinkDetectorForPHP was renamed to phpHyperlinkDetector.

8. Limitations

Debugger extension points are limited to use with Zend Debugger (phpDebugParametersInitializer, phpDebugHandlers, phpDebugMessages).

9. Open Questions and Debates

N/A