Design Document

PDT
0.7

API

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

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 and 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[1]. 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:

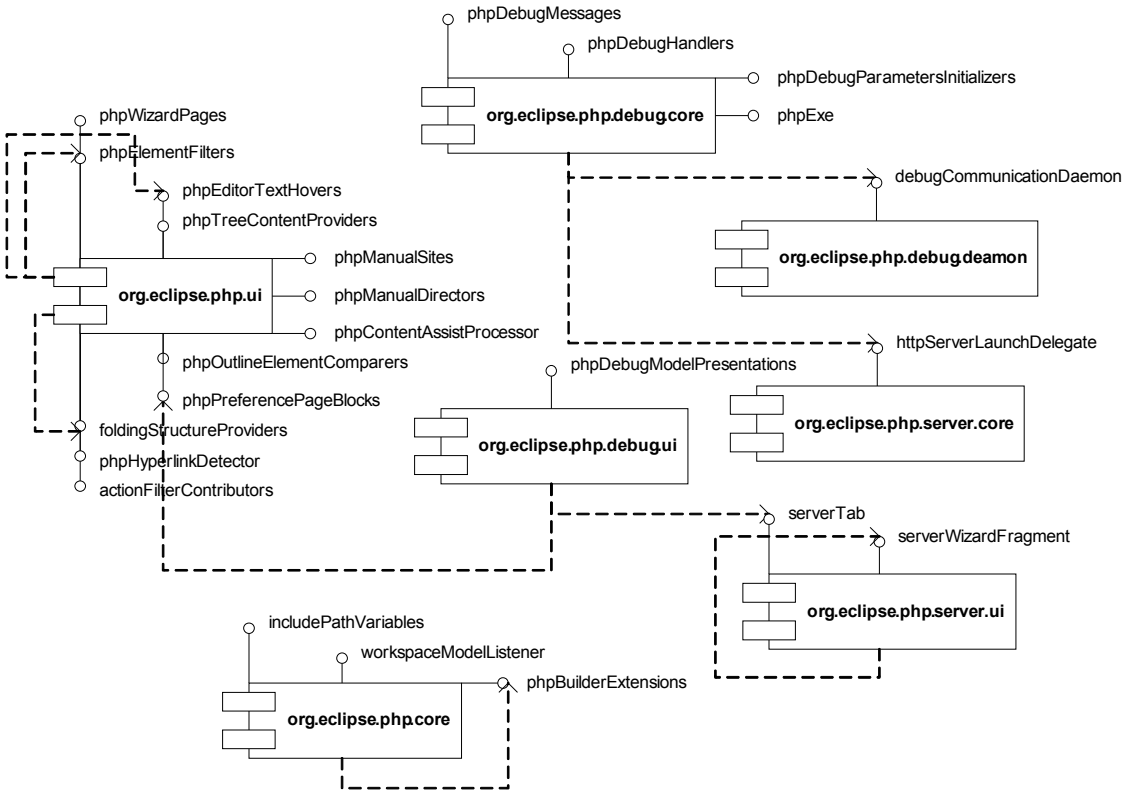


Figure 1 - PDT plugins dependencies

3.2Resources

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.

4.1Plug-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.

4.1.1PHP 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:

```
<!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:

```
<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:

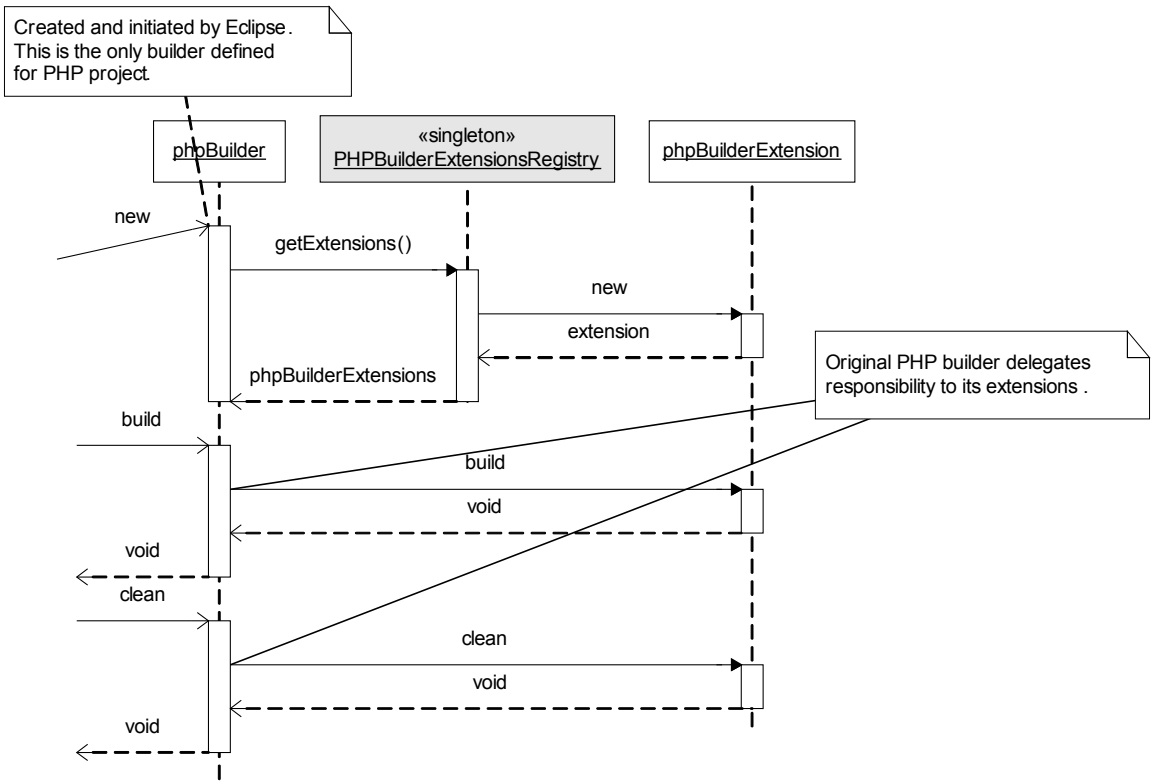


Figure 2 - PHP Builder Extensions

4.1.2Include 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:

```
<!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:

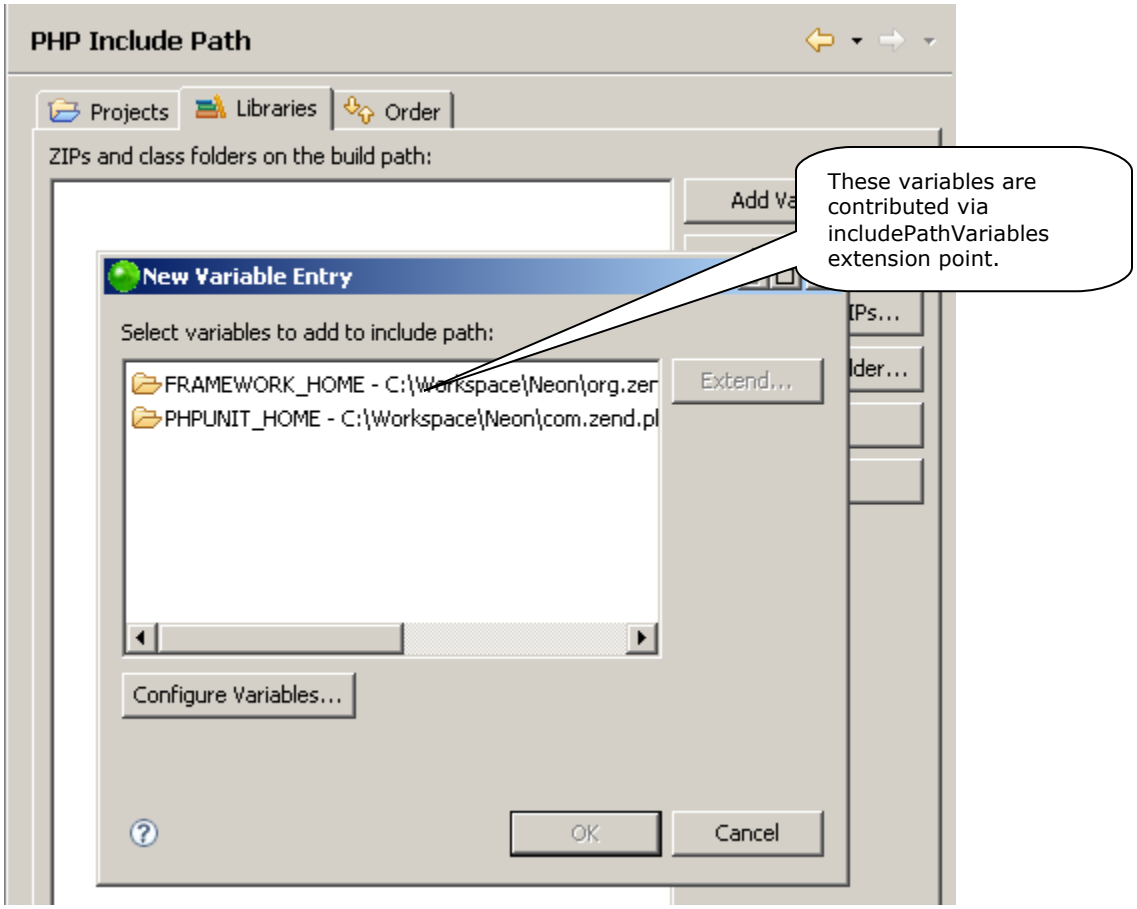
```
<extension point="org.eclipse.php.core.includePathVariables">
  <variable
      name="PEAR_LIBS"
      initializer="test.PearLibsVariableInitializer"/>

  <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:



4.1.3Workspace 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:

```
<!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

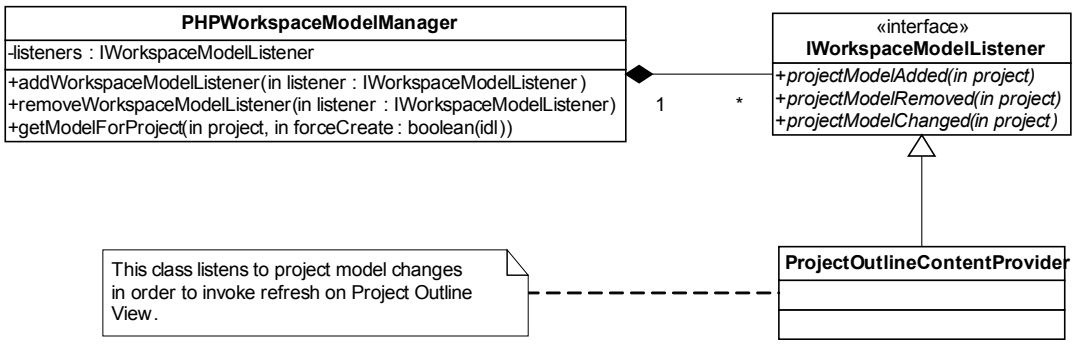
Declaration example:

```
<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:



4.2Plug-in org.eclipse.php.ui API

4.2.1PHP 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:

```
<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.

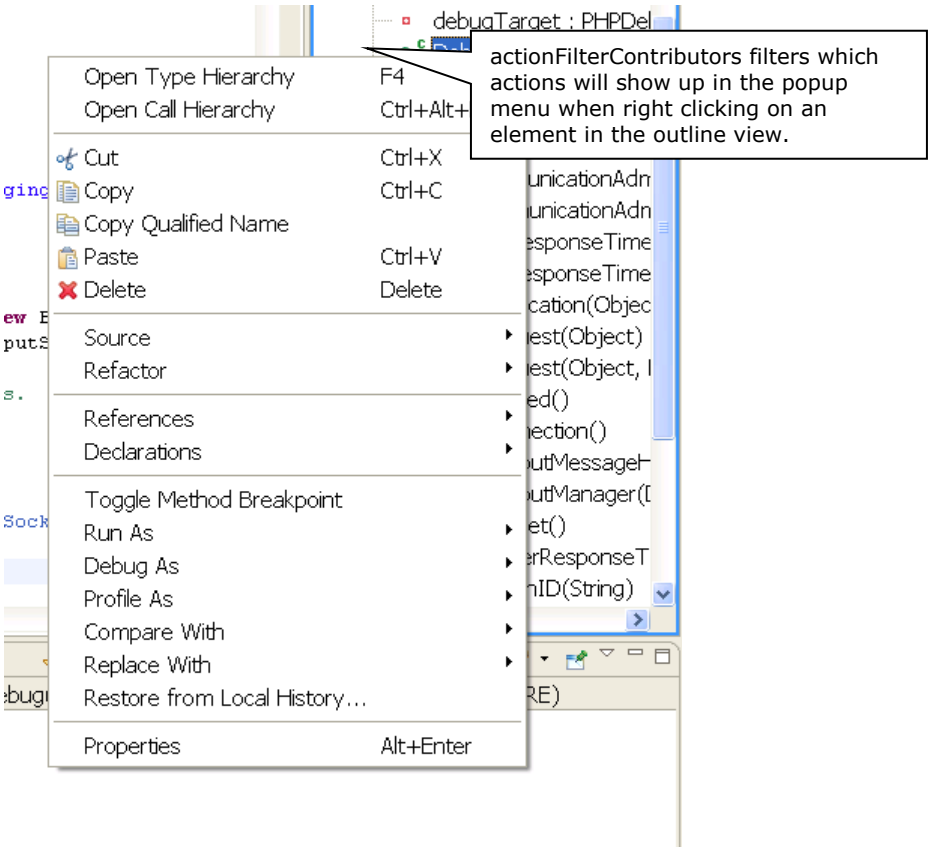
API Information:

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:

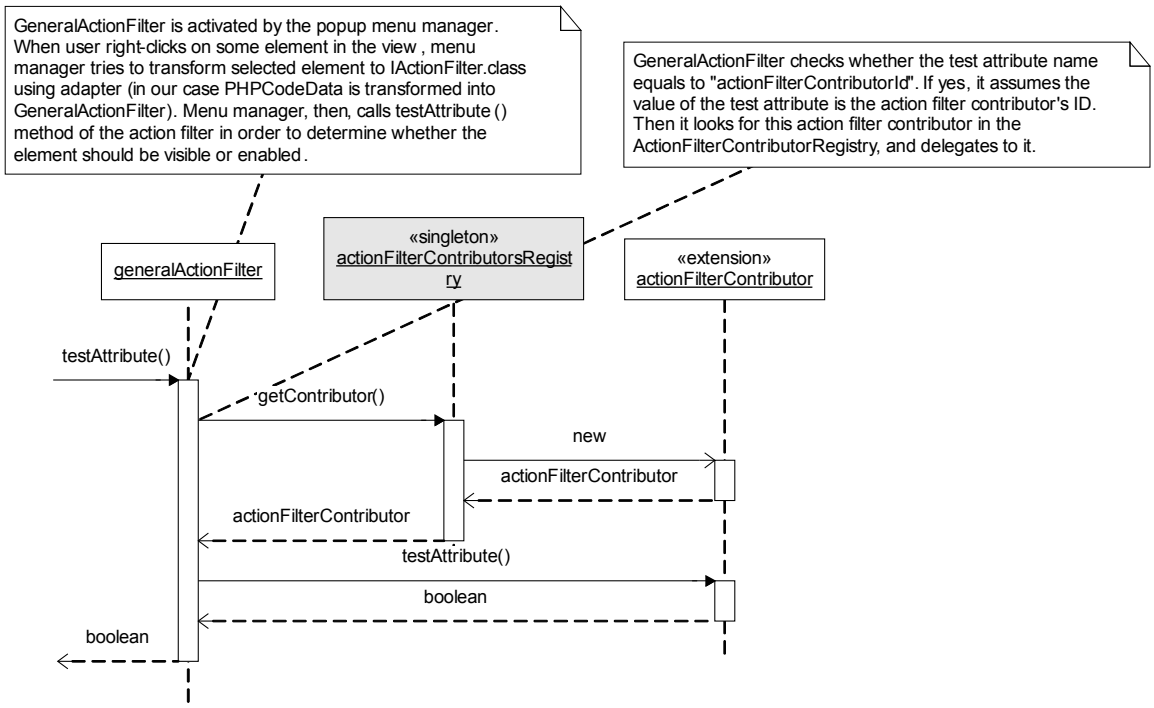


Figure 3 - Action Filter Contributor extension activation

4.2.2PHP 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:

```
<!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:

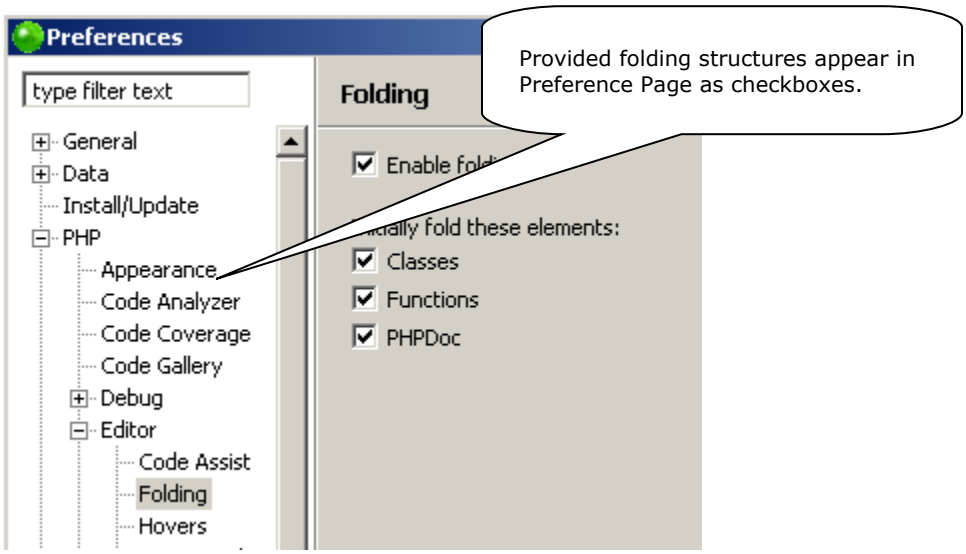
See `org.eclipse.php.internal.ui.folding.DefaultPHPFoldingStructureProvider` for an example.

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:


Preferences
type filter text
General
Data
Install/Update
PHP
Appearance
Code Analyzer
Code Coverage
Code Gallery
Debug
Editor
Code Assist
Folding
Hovers
Folding
Enable fold
ially fold these elements:
Classes
Functions
PHPDoc
Provided folding structures appear in Preference Page as checkboxes.

4.2.3PHP 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:

```
<!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

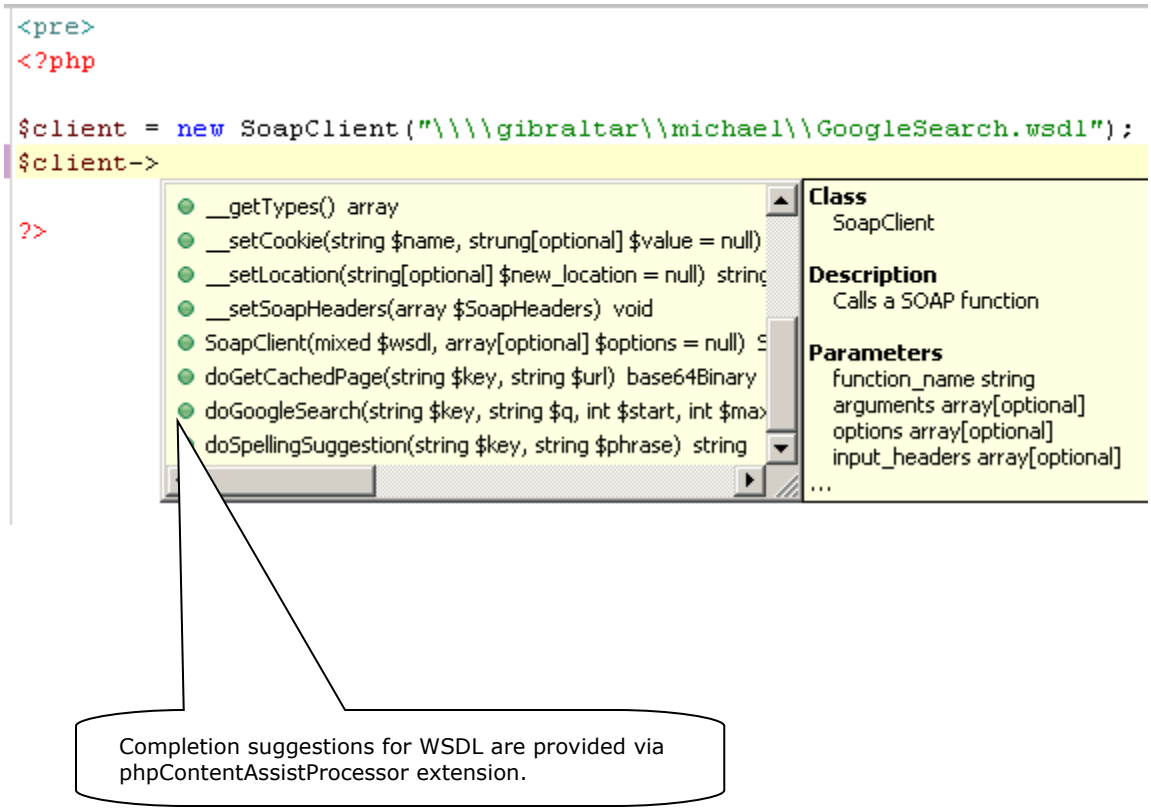
Declaration example:

```
<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:



Completion suggestions for WSDL are provided via phpContentAssistProcessor extension.

4.2.4PHP 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:

```
<!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:

```
<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:

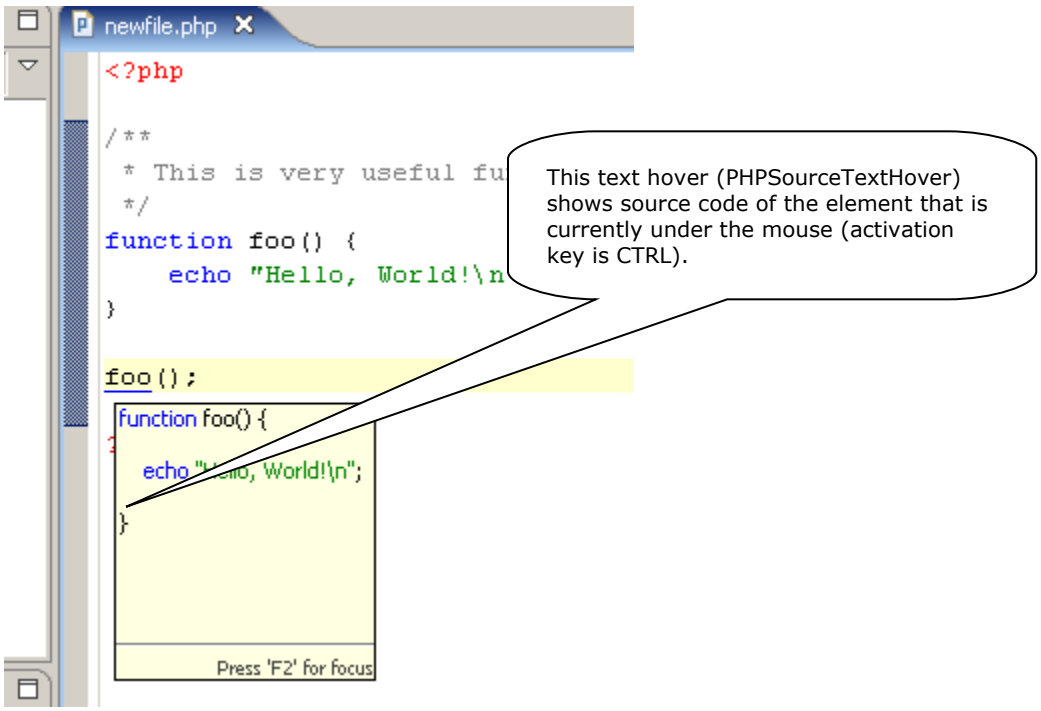
`org.eclipse.php.ui.editor.hover.ProblemHover` - used for displaying syntax error information.

`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:



4.2.5PHP 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:

```
<!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:

```
<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>
```

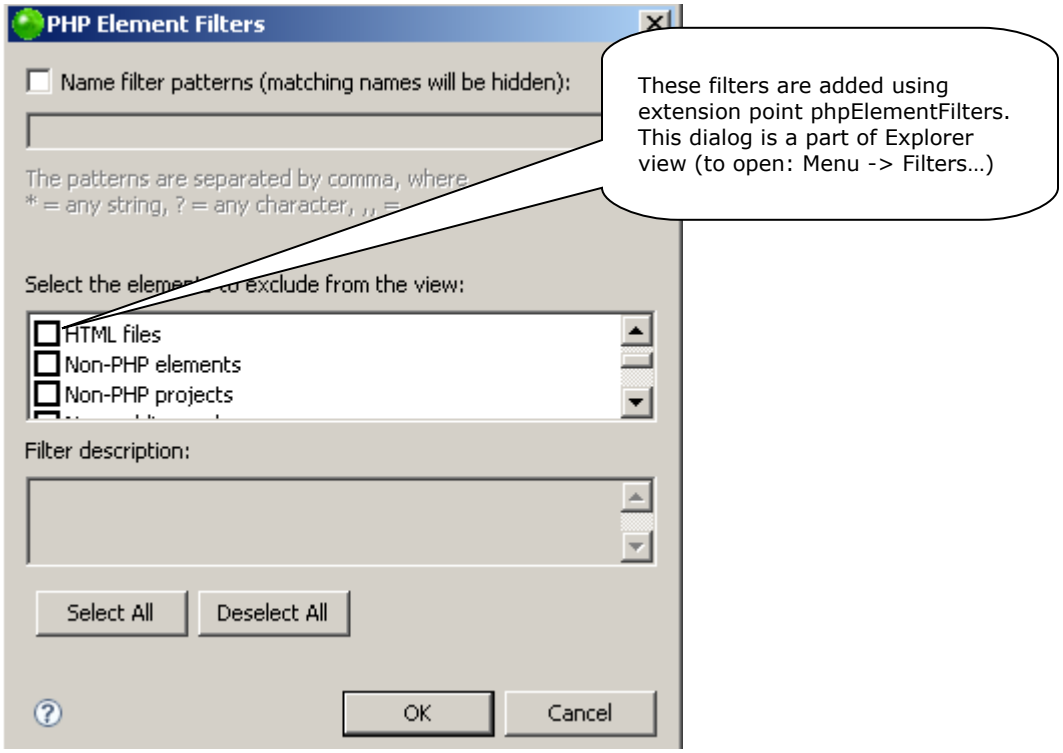
API Information:

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:



4.2.6PHP 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:

```
<!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:

```
<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.

4.2.7PHP Manual Director

Identifier: org.eclipse.php.ui.phpManualDirectors

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:

```
<!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:

```
<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:

See Default PHP Manual Director (`org.eclipse.php.ui.phpManualDirector`) defined in the PHP UI Plug-in.

4.2.8PHP 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:

```
<!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>
```

url - URL of the PHP Manual site, i.e.: http://www.php.net/manual/en/

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:

```
<extension point="org.eclipse.php.ui.phpManualSites">
   <site url="http://www.php.net/manual/en/"
      director="org.eclipse.php.ui.phpManualDirector"/>
</extension>
```

API Information:

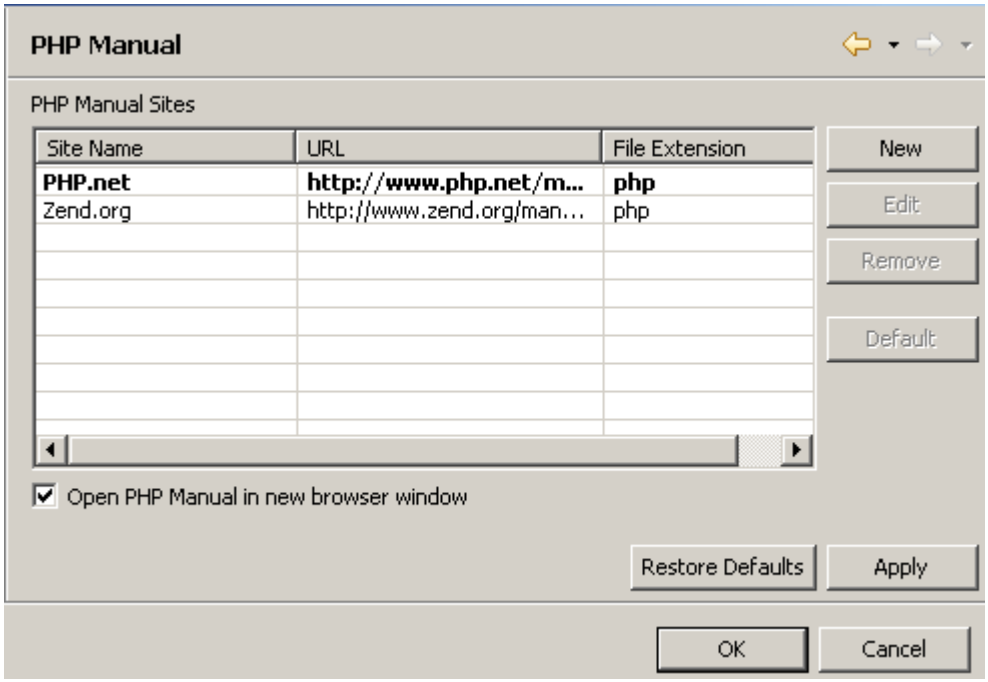
No API information, this extension point provides only configuration entries.

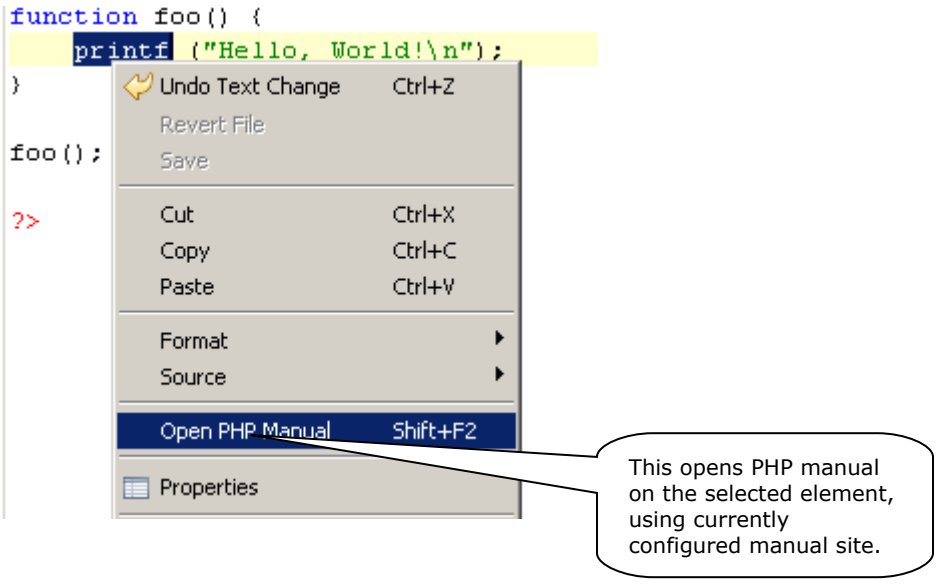
Supplied Implementation:

`org.eclipse.php.ui.phpDotNet` extension is contributed in PHP UI Plug-in, which defines a default PHP site (http://www.php.net).

Example of this extension point in action:

List of provided Manual sites can be seen in preference page.


PHP Manual
PHP Manual Sites
Site Name
URL
File Extension
PHP.net
http://www.php.net/m...
php
Zend.org
http://www.zend.org/man...
php
New
Edit
Remove
Default
Open PHP Manual in new browser window
Restore Defaults
Apply
OK
Cancel


function foo() {
printf ("Hello, World!\n");
}
foo();
?>
Undo Text Change Ctrl+Z
Revert File
Save
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Format
Source
Open PHP Manual Shift+F2
Properties
This opens PHP manual on the selected element, using currently configured manual site.

4.2.9PHP 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:

```
<!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:

```
<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:

```
<!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.

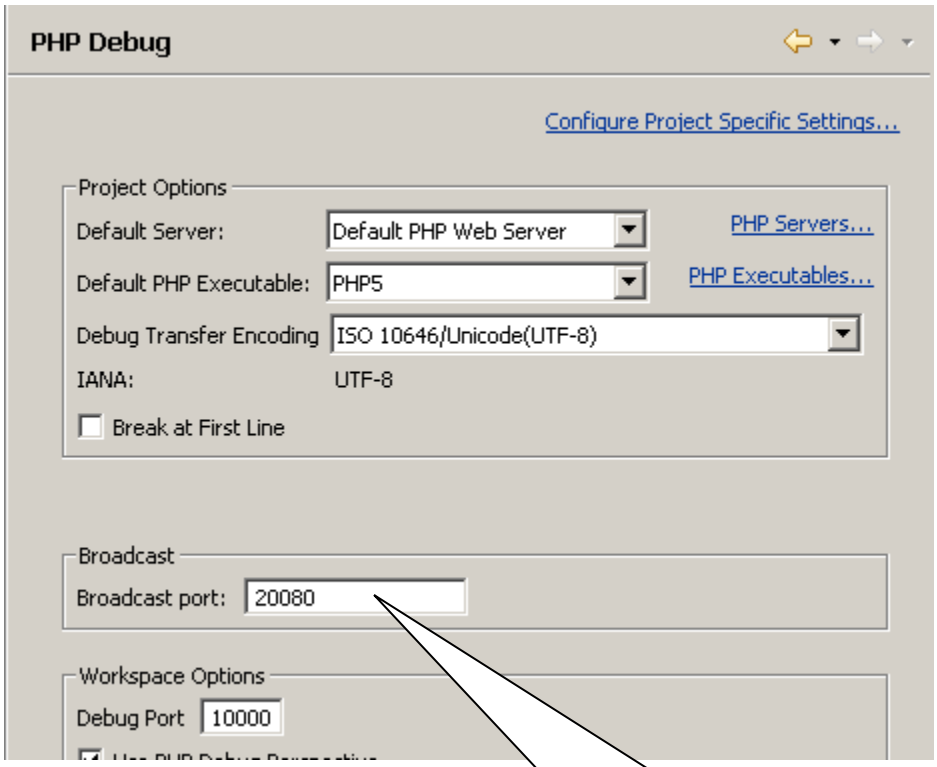
Declaration example:

```
<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.preferences.PhpDebugPreferen
cePage"
/>
</extension>
```

API Information:

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:

```
<!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:

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

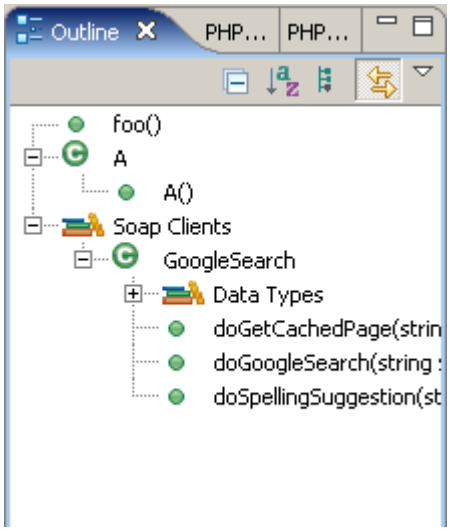
API Information:

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:


Outline
PHP...
PHP...
foo()
A
A()
Soap Clients
GoogleSearch
Data Types
doGetCachedPage(strin
doGoogleSearch(string
doSpellingSuggestion(st

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:

```
<!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:

```
<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.PHPProjectCreationWizar
   d"/>
</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.

4.3Plug-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:

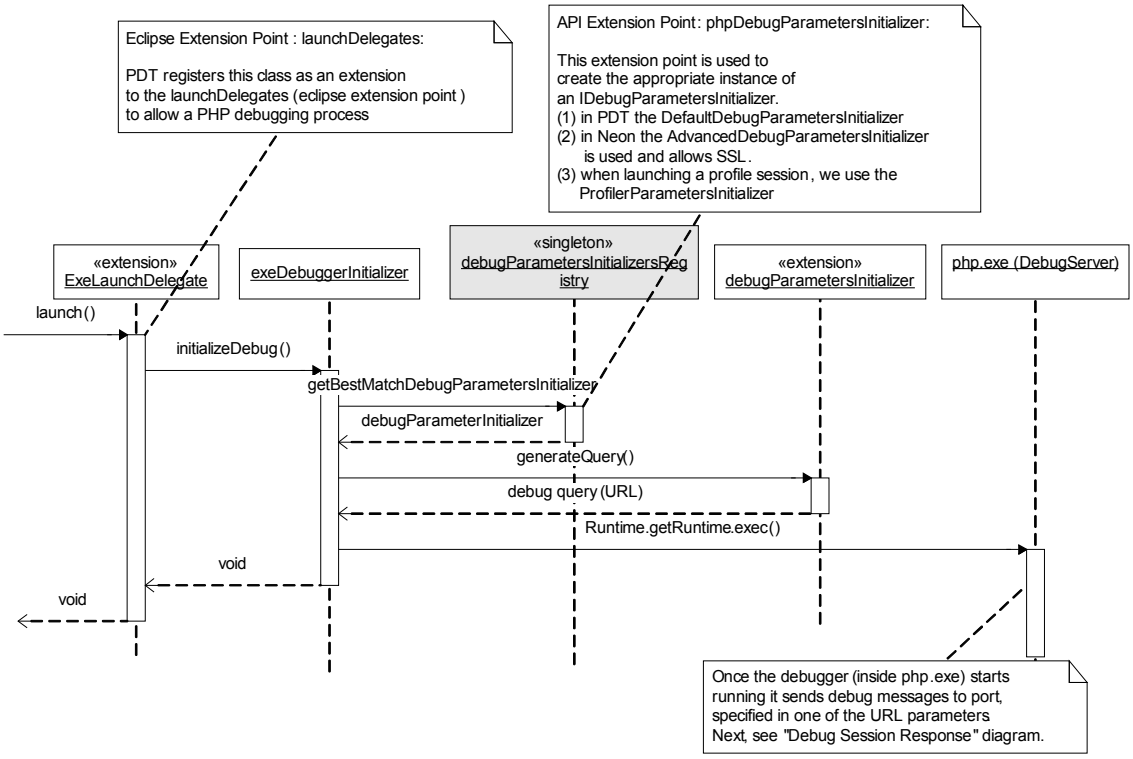


Figure 4 - Launch debug session using internal debugger

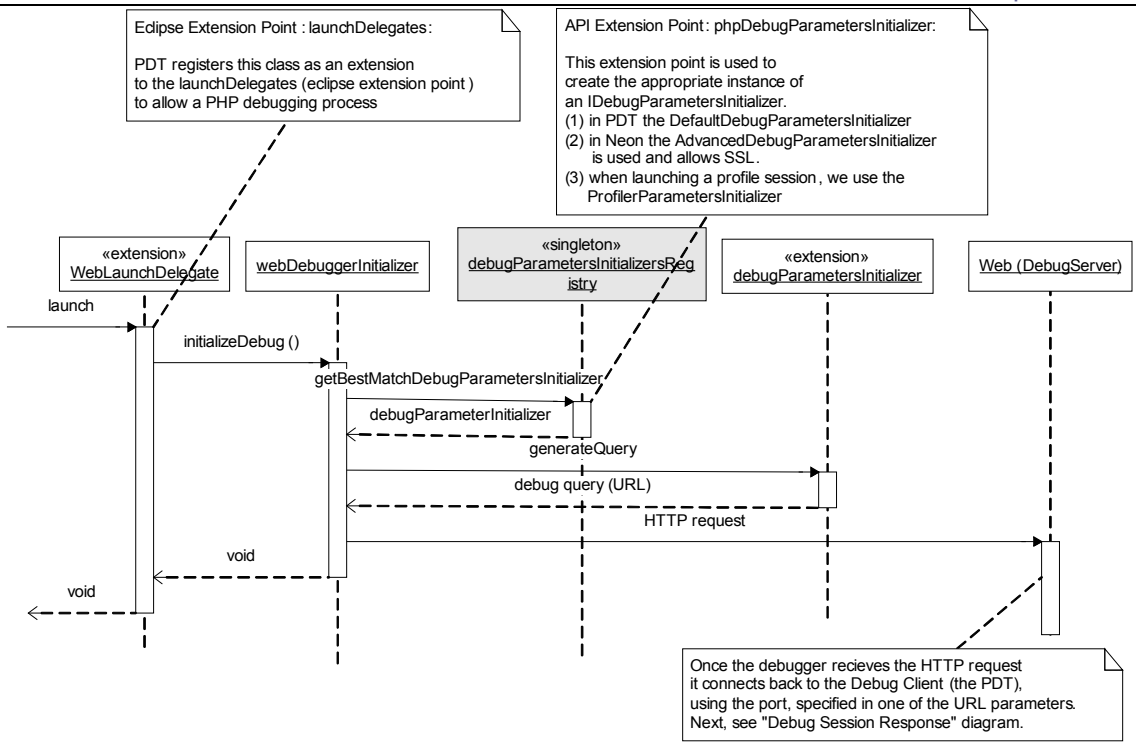


Figure 5 - Launch debug session using remote debug server

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 desirializing the debug message received from the debugger.

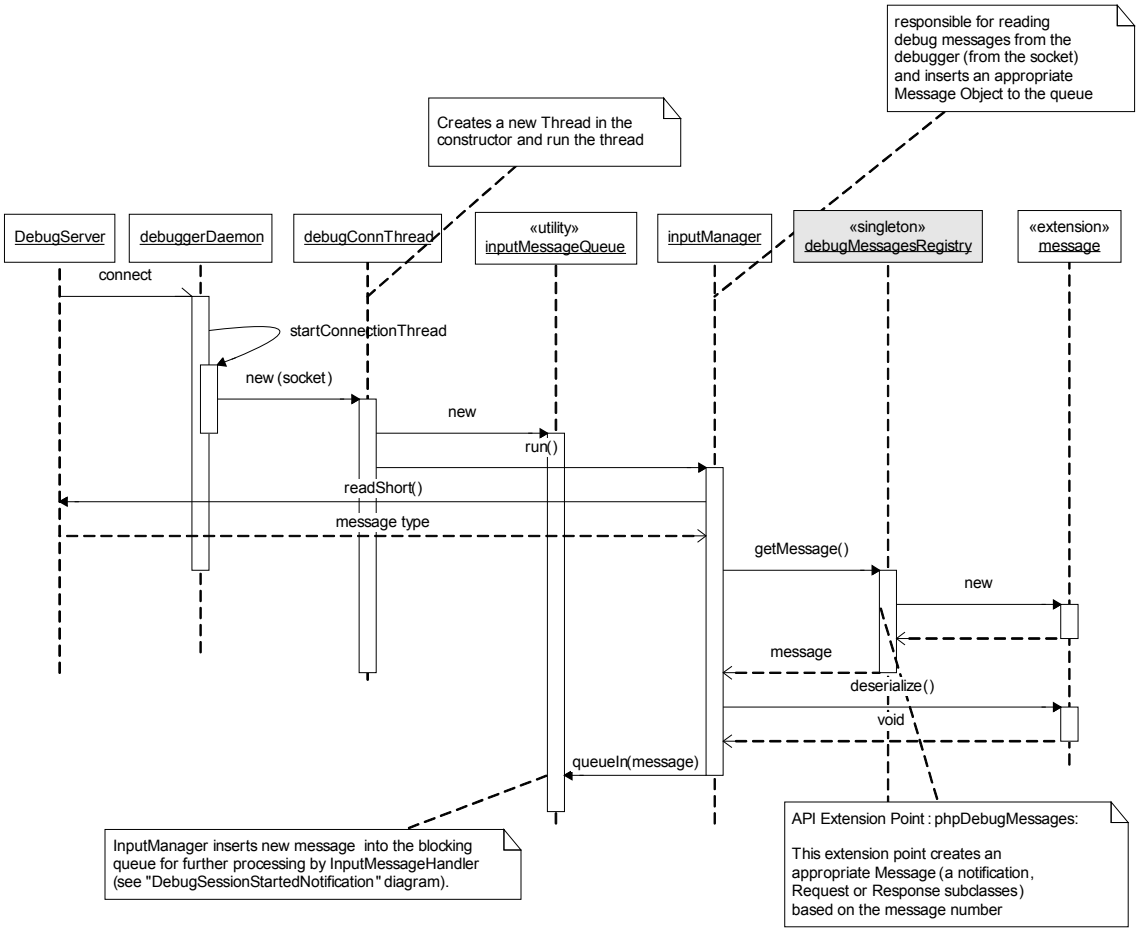


Figure 6 - process asynchronous request for starting new debug session.

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:

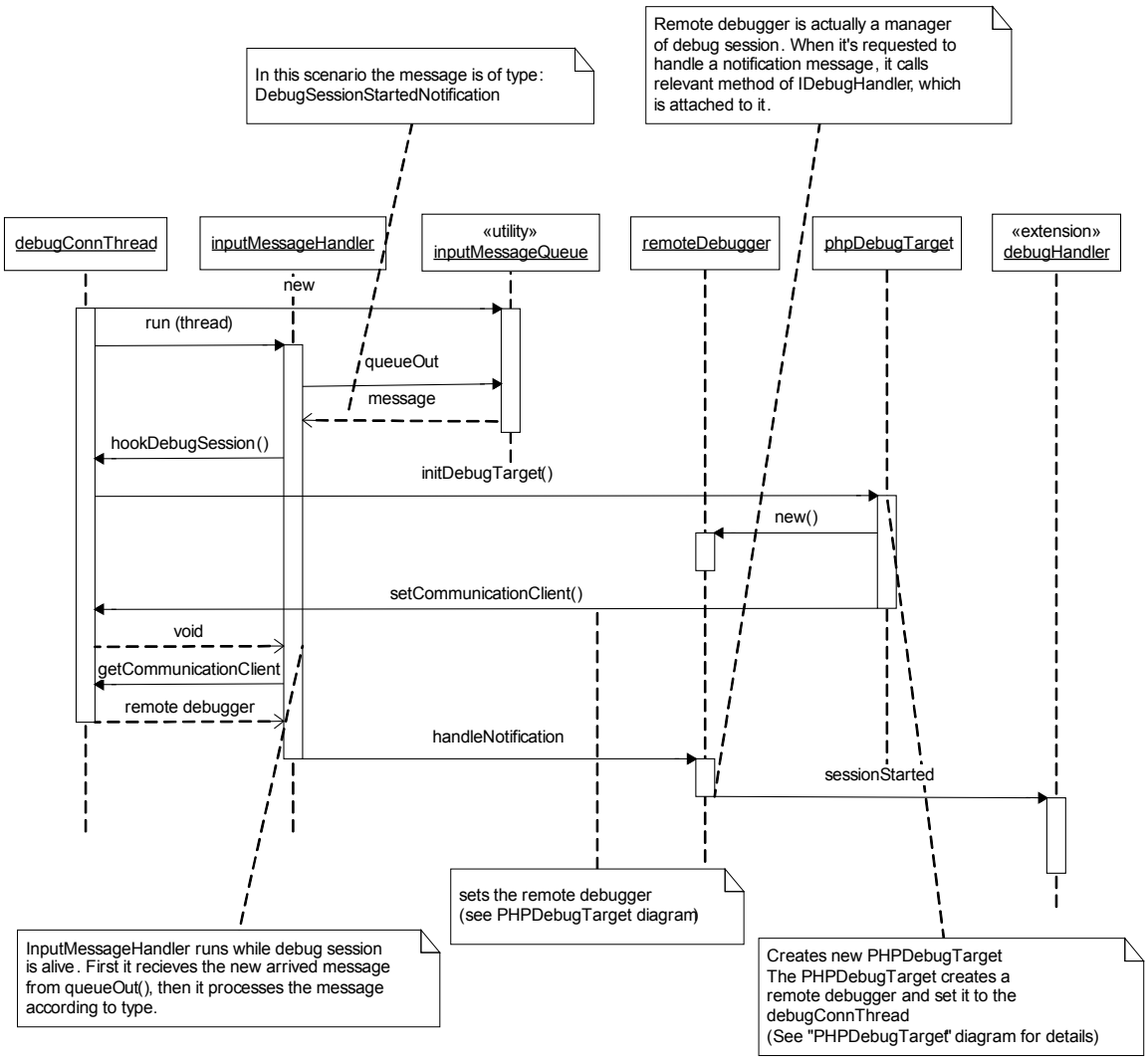


Figure 7 - processing DebugSessionStartedNotification message

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

The following sequence diagram shows the creating DebugHandler by the phpDebugTarget:

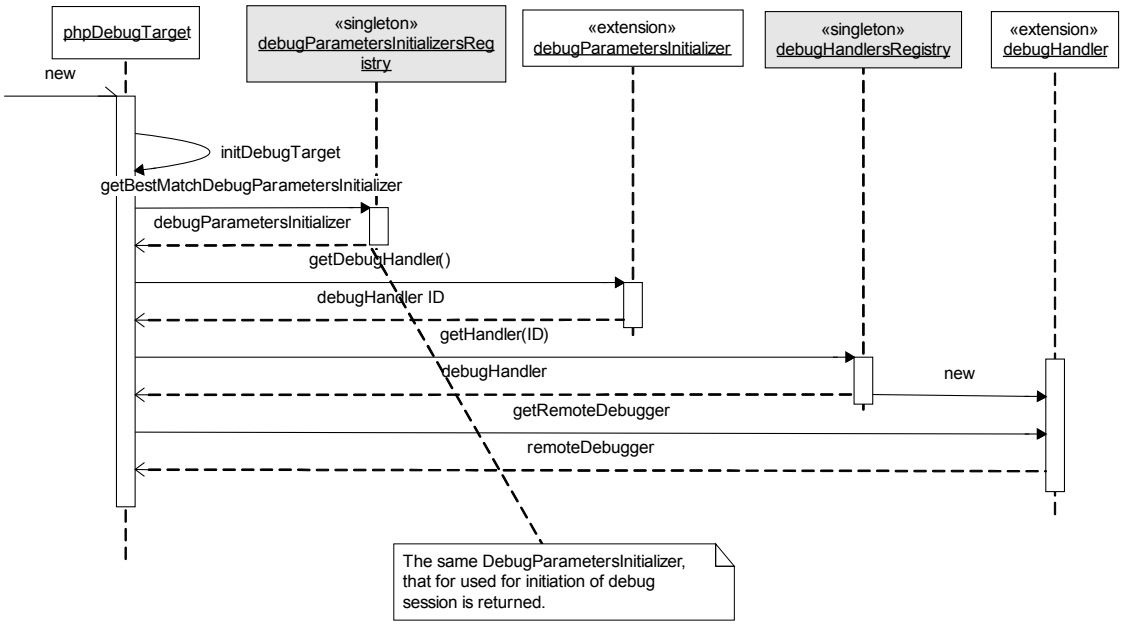


Figure 8 - creating DebugHandler

The following subsections elaborate on each of the extension point:

4.3.1PHP 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:

```
<!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:

```
<extension point="org.eclipse.php.debug.core.phpDebugParametersInitializers">
  <initializer
class="org.eclipse.php.debug.core.debugger.parameters.DefaultDebugParametersIni
tializer"
    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:

Look at `org.eclipse.php.debug.core.debugger.parameters.DefaultDebugParametersInitializer`.

Figure 3 and 7 (above) shows the usage of phpDebugParameterInitializer.

4.3.2PHP 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 recieved from PHP Debug Server. See `org.eclipse.php.debug.core.debugger.IDebugHandler`.

Configuration Markup:

```
<!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:

```
<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`.

4.3.3PHP 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:

```
<!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 `IDebugMesssage`.

handler - Handler class for this message that implements `IDebugMessageHandler`.

Declaration example:

```
<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`.

4.3.4PHP 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:

```
<!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:

```
<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:

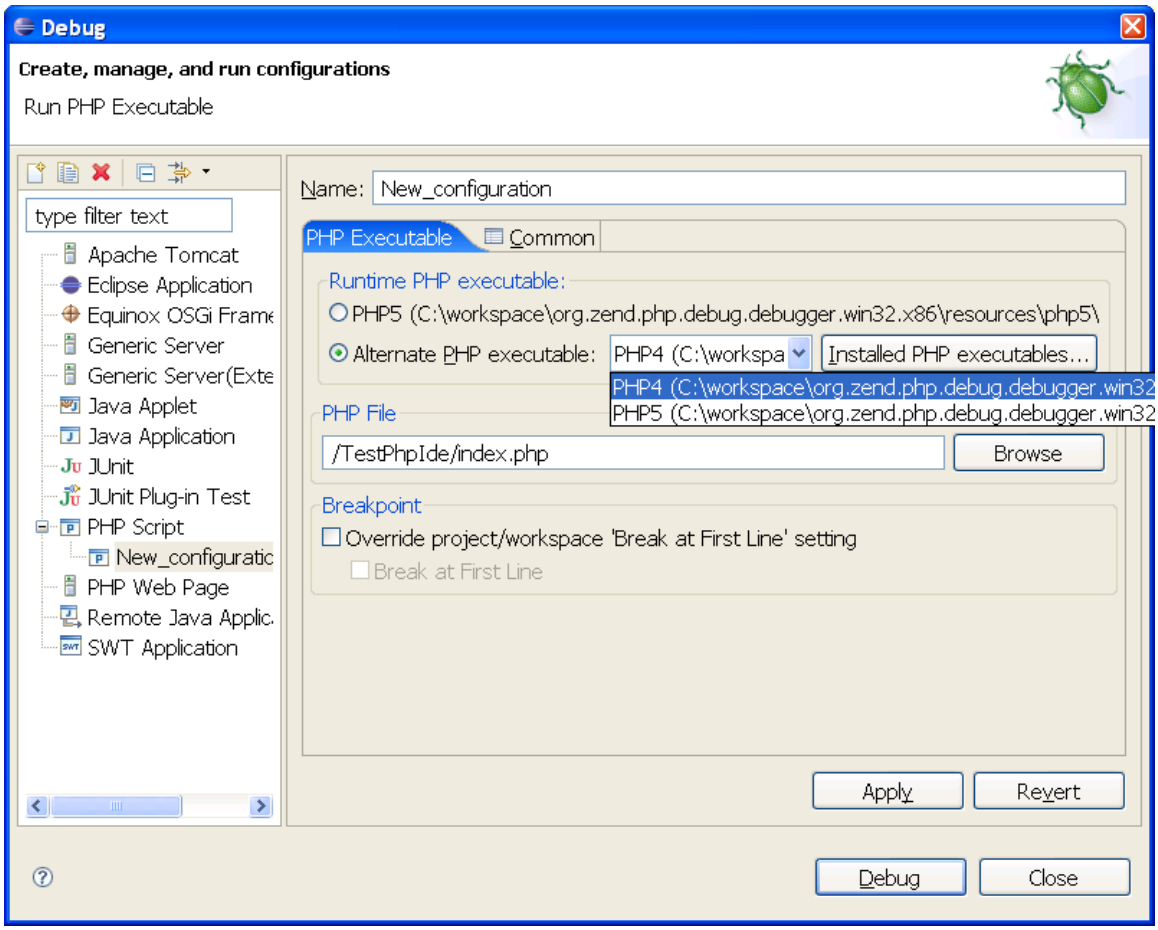


Figure 9 - selecting a PHP executable for debug session

4.4Plug-in org.eclipse.php.debug.deamon API

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

4.4.1Debug 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:

```
<!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:

```
<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`

4.5Plug-in org.eclipse.php.debug.ui API

4.5.1PHP 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:

```
<!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:

```
<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>
```

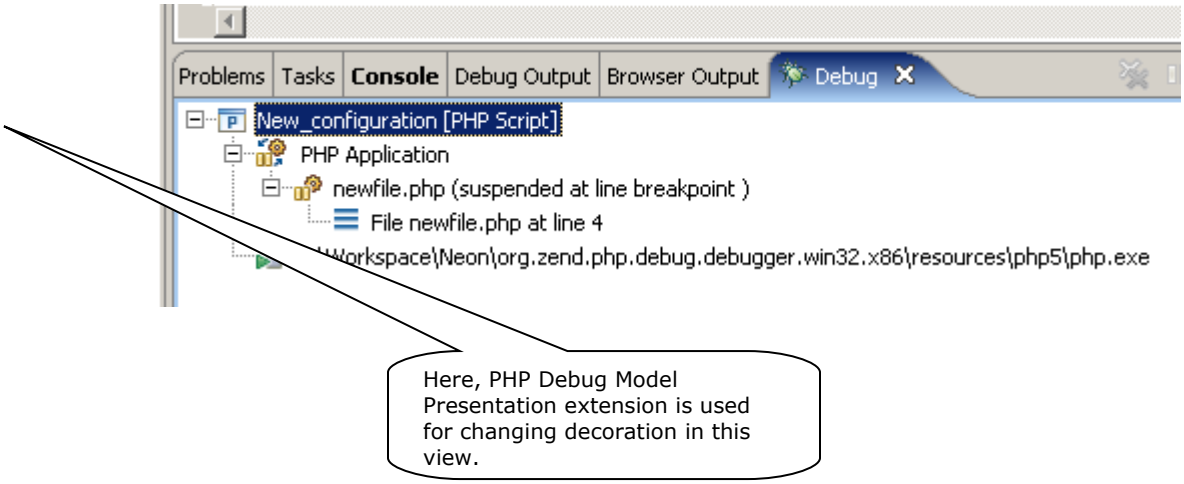
API Information:

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:



4.6Plug-in org.eclipse.php.server.core API

4.6.1httpServerLaunchDelegate

Type:

Description:

Example:

How to use:

4.7Plug-in org.eclipse.php.server.ui API

4.7.1serverTab

Type:

Description:

Example:

How to use:

4.7.2serverWizardFragment

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.

Test Item	Goal

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). 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)

2) Add/Remove Tapestry Project Nature (http://prod1.cleancode.com/spindle/infocenter/index.jsp?topic=/com.iw.plugins.spindle.docs/docs/tapestry_nature_work.html)

3) Toggle ANTLR plug-in nature (http://antlreclipse.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