

PDT Debugger Changes

This document describes the UI changes as well as the design changes that the PDT debugger infrastructure has gone through during the integration of the XDebug support.

To give a good understanding of the changes and their implications, this document will describe them from the UI to the design in this order:

1. Preferences UI changes.
2. Launch configurations.
3. Launch shortcuts.
4. Design changes and new extension points.

Preferences UI changes

PHP | Debug

The PHP | Debug preferences page has gone some changes to support the selection of the default workspace debugger and project-specific debugger (*figure 1*).

Also, it introduced a new table component that displays in a very simple way what are the installed debuggers and allow configuring some of their properties, such as the listening port (*figure 2*).

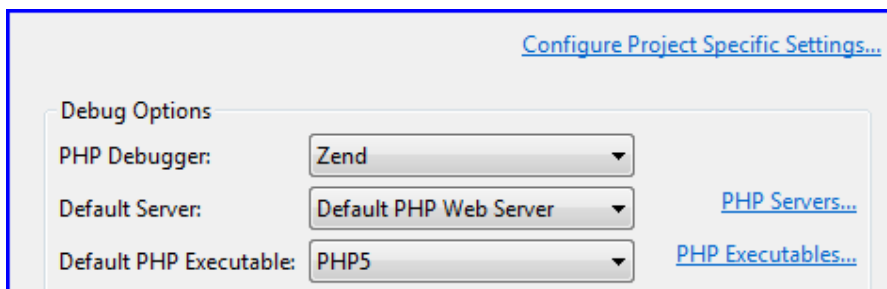


Figure 1: A selection of a default debugger type

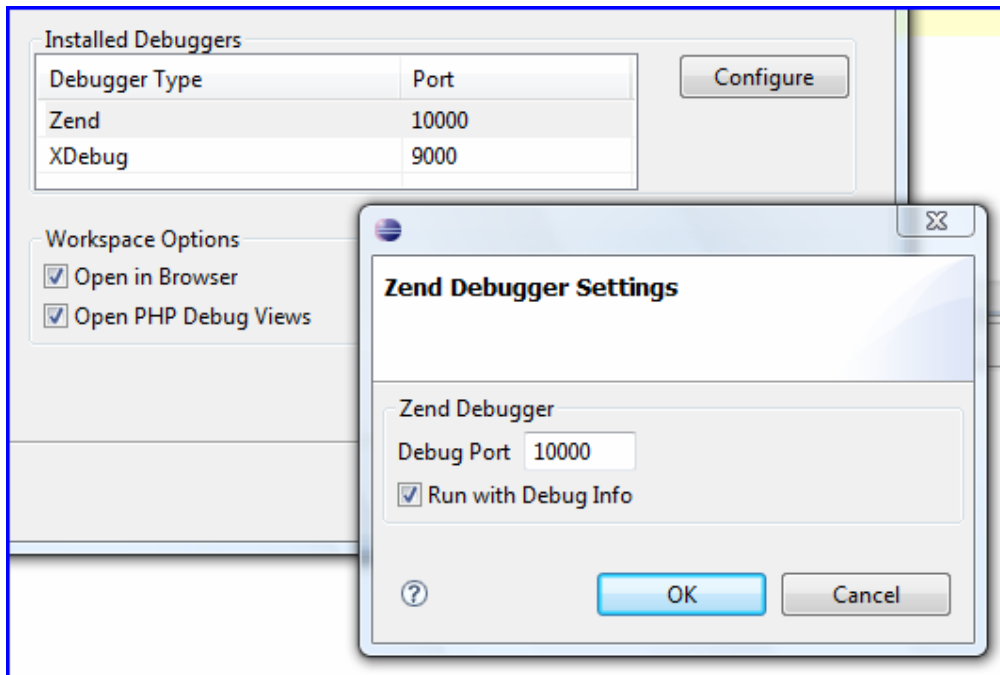


Figure 2: Setting the properties of an installed debugger using the debuggers table.

PHP | PHP Executables

The PHP Executables table now displays a new column for the 'Debugger Type' and has several default values, one for each debugger (figure 3).

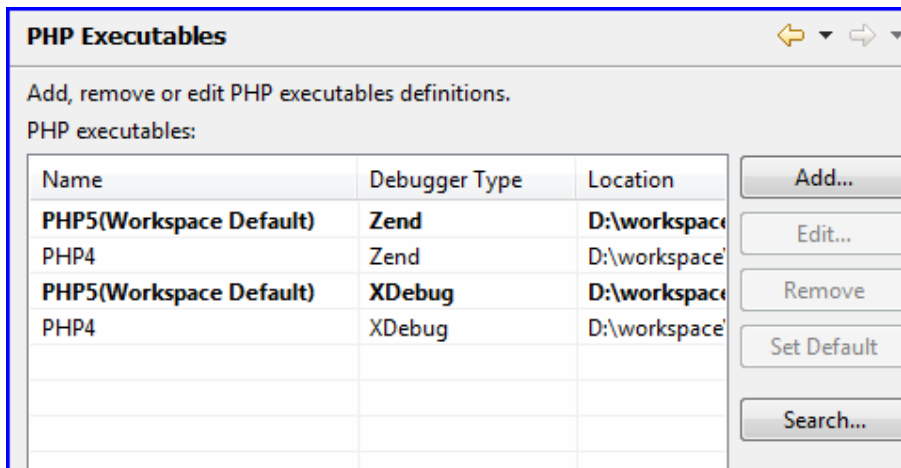


Figure 3: Executables table

There is a direct connection between the PHP Executables table and the PHP | Debug page. When you select the debugger type in the PHP | Debug, the displayed list of 'Default PHP Executable' displays only the ones that are relevant to the selected debugger type.

Adding new PHP executables is now done with an additional field that indicates the debugger that this executable will be used by (see figure 4)

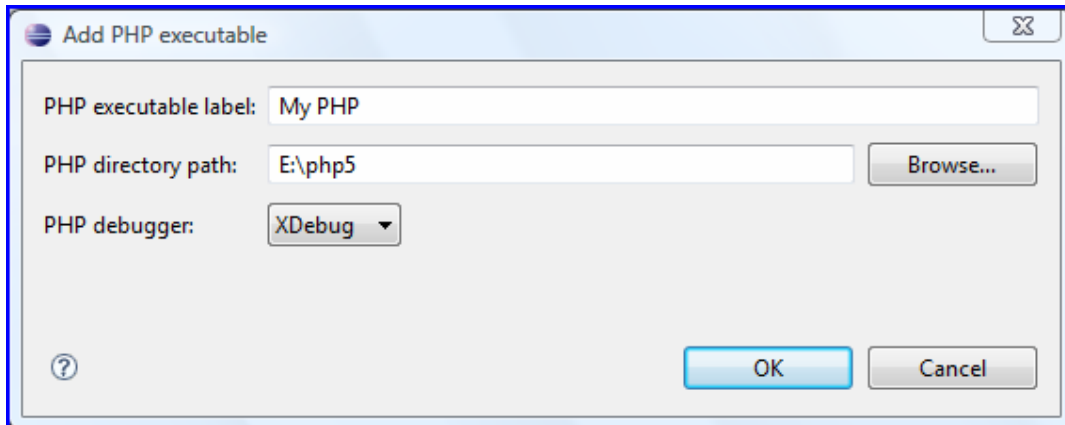


Figure 4: Selecting the debugger type when adding a new PHP executable.

Launch Configurations

PHP Script

The PHP Script launch configuration provides a new debugger type selection control. Selecting a specific debugger displays only the appropriate PHP executables, just like in the PHP | Debug page described previously.

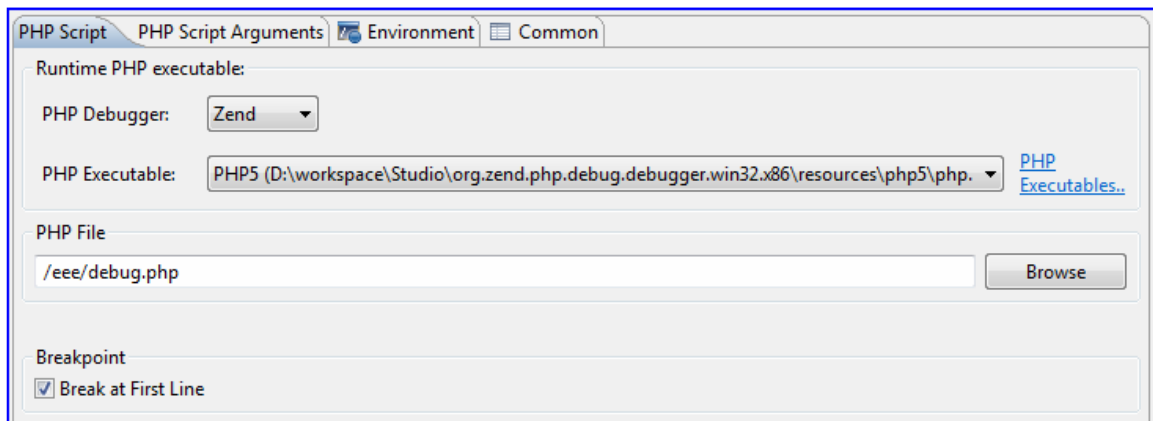


Figure 5: PHP Script launch configuration tab.

Creating new PHP Scripts configurations takes the default values from the workspace settings that can be changed through the PHP | Debug preferences page.

The launch configuration has also two new tabs additions: The 'PHP Script Arguments' and the 'Environment' tabs.

In the argument tab, a user can enter additional arguments that will be passed to the created PHP process when the launch is executed by Debug / Run.

The Environment tab can edit or add environment variables that will be used when debugging or running the script.

Note that in case none is entered, the native environment variables will be used (+some that are needed to a successful execution of the script).

PHP Web Page

The PHP Web Page launch configuration also has an additional debugger selection option. This selection option, for now, only affects the listener thread that will take over the debug session that arrives from the remote debug server (see figure 6 for the new addition).

Also, the 'Break at First Line' was moved from the 'Advanced' tab to the 'Server' tab.

At this stage, the 'Advanced' tab is only affective for Zend debugger.

The screenshot shows the 'WebDebug' launch configuration dialog with the 'Server' tab selected. The 'Server Debugger' is set to 'XDebug'. The 'PHP Server' is set to 'Default PHP Web Server'. The 'File / Project' section shows the file path '/eee/debug.php' with a 'Browse' button. The 'Breakpoint' section has a checkbox for 'Break at First Line'. The 'URL' section has a checkbox for 'Auto Generate' which is checked, and the URL is set to 'http://localhost/debug.php'.

Name: WebDebug

Server Advanced Common

Server

Server Debugger: XDebug

PHP Server: Default PHP Web Server New Configure...

File / Project

/eee/debug.php Browse

☐ Publish files to Server

Publish To: \

Breakpoint

☐ Break at First Line

URL

☒ Auto Generate

URL: http://localhost/debug.php

Figure 6: PHP Web Page launch configuration tab.

Launch Shortcuts

In order to provide a clear and easy launches directly from the editor or the PHP explorer, the PDT continue to provide launch shortcuts directly from the context menus of these components.

We maintain this behavior for multiple debuggers support in and when launching a session using a launch shortcut the PDT takes the project-specific settings for the current resource (or the workspace settings if there are no specific settings) and creates a launch configuration that already contains the default debugger type for the resource.

Note: *The next sections are directed to developers that are interested in expanding the PDT debug functionalities*

Design Changes

Supporting multiple debuggers for the PDT required some major conceptual changes in the way we hook the debuggers to the system.

The main motivation was to integrate XDebug support into the code-base of the PDT; however, we did that by enabling the addition of other debuggers that might be available in the future.

The following bullets will describe all the major changes that were made to support multiple debuggers in general and XDebug in particular.

Multiple debuggers support

New extension point

A new `org.eclipse.debug.core.phpDebuggers` extension point was introduced.

This extension point allows a definition of a debugger by adding its name, id and its debugger configuration instance, which is an implementation of `IDebuggerConfiguration` interface.

The `IDebuggerConfiguration` interface is the main supplier to some of the debugger attributes and it supplies basic attributes, such as the debug port, the name and the debugger's id.

But the real strength of this debugger configuration is by supplying the UI implementation for the PHP | Debug preferences page when a user wants to modify the specific debugger's attributes.

Also, this debugger configuration supplies the fully qualified class names to use as launch delegates when a launch for the specific debugger was identified (see explanation about the proxy launch delegate a few sections below).

To make this configuration class even more flexible, we added an option to insert / extract any String attributes programmatically.

A convenient base implementation of this interface is the `AbstractDebuggerConfiguration` class (which is also the one required by the extension point).

Modified extension points

The `org.eclipse.debug.core.phpExe` extension point was modified to include a debugger id for any of the registered PHP executables.

Debug daemons initialization

Created an `AbstractDebuggerCommunicationDaemon` that supplies basic implementation for every registered debugger daemon.

For now, this is the base class for Zend's `DebuggerCommunicationDaemon` and XDebug's `XDebugCommunicationDaemon`.

The `DaemonPlugin` is no longer initiating the daemons. This responsibility was moved to the `PHPDebugPlugin` that initiate them while starting.

Also, the `DaemonPlugin` can now support initiation of a specific debugger by receiving its identification string.

Using the PHPLaunchDelegateProxy

The PHPLaunchDelegateProxy was previously used only for the PHP executables launch delegates. We now extended it use to the PHP Web Page delegates and this gives us the power to select the appropriate launch delegate at runtime.

As mentioned above, the debugger extension point supplies an IDebugConfiguration instance that holds the delegates class names for any launch type (script / web page).

Once a launch is triggered, the proxy acquires the appropriate delegate by extracting the `CONFIGURATION_DELEGATE_CLASS` from the launch configuration and then initiates an instance of this class by reflection.

This `CONFIGURATION_DELEGATE_CLASS` attribute is inserted to the launch configuration from the different launch tabs or launch shortcuts after consulting the debug configuration for the appropriate class name for the launch type.

XDebug related changes

The initial implementation of the XDebug support was implemented by Dave Kelsey and most of it was left intact. However, some of the classes were modified according to the latest required changes.

- Added a new `XDebugCommunicationDaemon` and hooked the XDebug daemon to the daemons extension point. This daemon actually invokes a new `DBGpSession` instance for every debug session.
- The XDebug daemon starts the listen thread when the PDT starts.
- Used one common source lookup classes for the XDebug and the Zend debugger.
- Made the XDebug executable launch delegate to support the `IniModifier` class which enables more flexibility when dealing with php.ini's.
- Added the XDebug debugger extension and it debugger definition to the phpDebuggers extension point.
- Removed the timeout directive from the initial XDebug implementation.

20 August 2007

- Added 'Publish' capabilities to the XDebug web page launch delegate.
- Added the 'PHP script argument' and 'Environment' tabs to the general PHP Script launch tabs.
- Use the PDT's Web Page launch configuration tab with the XDebug web page launch delegate.

Other changes

A package structural change was made to differentiate the core Zend debugger code and the core XDebug code, and you can now see that there are new zend and xdebug packages containing only the related Zend and XDebug classes accordingly.

Summery

The PDT debugger have evolved to support several debuggers, and the first one that was added to its code base is the XDebug.

Many thanks to Dave Kelsey who contributed the XDebug code and will continue to contribute to this project in the future.

Shalom Gibly

PDT Team

Zend Technologies