Zero-Configuration Debugging

In case of *zero-configuration debugging*, you do not need to create any debug configuration. Instead, you open the starting page of your PHP application in the browser manually, and then activate the debugging engine from the browser, while IntelliJ IDEA listens to incoming debugger connections.

To enable starting and stopping the debugging engine from the browser manually, you need to set a special `GET`/`POST` or `COOKIE` parameter. You can do it manually in the `php.ini` configuration file or generate the *Start Debugger/Stop Debugger* bookmarklets on the toolbar of your browser. These bookmarklets provide control over the debugger cookie, through them you will activate and deactivate the debugger.

For more details about setting the parameters manually, see Starting the Debugger for *Xdebug* and Zend Debugger GET Request Parameters for *Zend Debugger*.

You can also specify the scripts requests to which you want IntelliJ IDEA to ignore during debugging. This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the *skipped paths* list. You can also group such scripts into folders and add these folders to the "ignore list".

On this page:

Preparing the debugging engine

Before you start debugging, make sure that you have a debugging engine installed and configured properly. IntelliJ IDEA supports debugging with two most popular tools: XDebug and Zend Debugger. These tools cannot be used simultaneously because they block each other. To avoid this problem, you need to update the corresponding sections in the `php.ini` file. To find out which `php.ini` file is active, create and run a test file with `phpinfo()`, then search for the *Loaded Configuration File*.

For more information on configuring debugging engines, see Configuring XDebug, Configuring Zend Debugger, http://confluence.jetbrains.com/display/PhpStorm/Xdebug+Installation+Guide , and http://confluence.jetbrains.com/display/PhpStorm/Zend+Debugger+Installation+Guide.

Setting the breakpoints

Breakpoints are source code markers used to trigger actions during a debugging session. Typically, the purpose behind setting a breakpoint is to suspend program execution to allow you to examine program data. However, IntelliJ IDEA can use breakpoints as triggers for a variety of different actions. Breakpoints can be set at any time during the debugging process. Your breakpoints don't affect your source files directly, but the breakpoints and their settings are saved with your IntelliJ IDEA project so you can reuse them across debugging sessions.

1. Place the caret on the desired line of the source code.

   Breakpoints can be set in the PHP context inside `*.php`, `*.html`, and files of other types. Only executable lines are valid locations for line breakpoints. Comments, declarations, and empty lines are not valid locations for the

2. Do one of the following:
   - Click the left gutter area at a line where you want to toggle a breakpoint.
   - On the main menu, choose **Run | Toggle Line Breakpoint**.
   - Press `Ctrl+F8`.

Generating the Start Debugger/Stop Debugger bookmarklets

These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.

- Enable the *Bookmarks toolbar* in your browser by doing one of the following depending on the browser type:
  - In *Firefox*, choose **View | Toolbar | Bookmarks Toolbar**.
  - In *Chrome*, choose **Bookmarks | Show bookmarks bar**.
- Open the Project Settings and click **PHP**. Then click **Debug** under the **PHP** node.
- On the **Debug** page, that opens, click the **Use debugger bookmarklets to initiate debugger from your favorite browser** link.
- On the Zend Debugger & XDebug bookmarklets page that opens, check the debugging engine settings and click **Generate**. The bookmarks for listed debugging-related actions are generated.
- Drag the generated links to the bookmark toolbar in your browser.

Initiating a debugging session and examining the suspended program

1. Generate the Start Debugger/Stop Debugger bookmarklets, if you have not done it yet. These bookmarklets will appear on the toolbar of your browser. They provide control over the debugger cookie, through them you will activate and deactivate the debugger.
2. Set the breakpoints, where necessary.
3. Toggle the **Start Listen PHP Debug Connections** button so it changes to . After that IntelliJ IDEA starts listening to the port of the debugging engine used in the current project. Ports for debuggers are set at the IntelliJ IDEA level in the Debug dialog box (**File | Settings | PHP | Debug**).
4. Open the starting page of your application in the browser.
5. To activate the debugging engine from the browser, choose the **Start debugger** bookmark.
6. Re-load the current page (the starting page of the application).
7. Switch to IntelliJ IDEA.
8. As soon as the debugger suspends on reaching the first breakpoint, examine the application by analyzing *frames*. A *frame* corresponds to an active method or function call and stores the local variables of the called method or function, the arguments to it, and the code context that enables expression evaluation. All currently active frames are displayed on the **Frames** pane of the Debug tool window. where you can switch between them and analyze the information stored therein in the **Variables** and **Watches** panes. For more details, see the section Examining Suspended Program.
9. Continue running the program and examine its frames as soon it is suspended.
   - To control the program execution manually, step through the code using the commands under the **Run** menu or toolbar buttons: **Step Into** (`F7`), **Step Out** (`Shift+F8`), **Step Over** (`F8`), and others. For more details, see Stepping Through the Program.
   - To have the program run automatically up to the next breakpoint, resume the session by choosing **Run | Resume Program** or pressing `F9`

Specifying scripts to skip requests to

This approach can be useful, when your application contains scripts that use AJAX. Suppose you have a `menu-ajax-script.php` that "reloads" a part of your web page. This script works properly so you do not need to debug it. However, this script is still requested during the debugging session. To have incoming connections to this script ignored, add the `menu-ajax-script.php` script to the *skipped paths* list. You can also group such scripts into folders and add these folders to the "ignore list".

1. Open the Project Settings, click **Debug** under the **PHP** node, then click **Skipped Paths**.
2. On the Skipped Paths page that opens, configure an "ignore list" of scripts and folders with scripts not to be invoked if IntelliJ IDEA receives incoming connections to them.
   - To add a new entry to the list, click the **Add** button + or press `Alt+Insert`. Then click the **Browse** button ... and in the dialog box that opens choose the file or folder to skip connections to.
   - To remove an entry from the list, select it and click the **Remove** button — or press `Alt+Delete`. The script will be now executed upon receiving requests to it.
3. To have IntelliJ IDEA inform you every time it receives a request to a script to be skipped, select the **Notify about skipped paths** check box.

See Also

Procedures:

- PHP Debugging Session
- Creating and Editing Run/Debug Configurations

- Debugging PHP Applications
- Debugging

Reference:

- Run/Debug Configuration: PHP Web Application
- Servers

External Links:

- http://confluence.jetbrains.com/display/PhpStorm/Zero-configuration+Web+Application+Debugging+with+Xdebug+and+PhpStorm

Web Resources:

- Developer Community