Native Xdebug Path Mapping

Introduction

Currently Xdebug only operates and handles system-local file names. It only sees names as they are on the system that Xdebug/PHP runs on. Many IDEs and debug adapters (IDEs) employ a “path mapping” feature, to map Xdebug/PHP-local paths to the filesystem on which IDE users edit files. This allows for IDEs to set breakpoints to the paths that Xdebug sees after translating them from local paths.

To make this terminology more clear, from now on I will refer to “remote paths” as the paths and file names that Xdebug sees on the system that it runs on, and “local paths” as the paths and file names in the environment where the IDE runs.

An example: Xdebug runs on a staging server, where an application is located in the directory `/var/www/staging.example.com`. These same files are equivalent to local paths on the system where the IDE can edit them in `/home/user/projects/example.com`. The **remote** path here is `/var/www/staging.example.com`, and the **local** path is `/home/user/projects/example.com`.

Although path mapping features for debugging in IDEs are a sensible solution, there are a few situations where this is not always enough:

- Some applications in PHP, most notably Neos/Flow, rewrite files during execution into a cache, which then do not match the right files that a developer would be editing. This is also a problem on local-only projects, where the application runs on the same host as where the code is edited.
- Setting up path mappings in an IDE can be challenging for developers of WordPress plugins.
- Templating systems often compile their template language down to PHP files with a different name, but also the generating PHP code will not match the line numbers with the lines in the template file.
- Xdebug can render links to paths whenever it displays a filename (in HTML) output, regardless of whether debugging is even enabled. Because IDEs are not involved here, their path mappings don’t apply.

Proposal

In order to make Xdebug more adoptable and usable for these cases where IDEs can’t always help, a new feature is needed which would allow the configuration of path, file, and line mappings solely through Xdebug.

Xdebug native path mappings can be configured through files.

Xdebug will scan for files ending in `.map` in the `.xdebug` directory when a script starts, and the Xdebug mode (set through the xdebug.mode setting or `XDEBUG_MODE` environment setting) is not `Off`.

Xdebug will check for the `.xdebug` directory in the grand-parent directory of the file that is being requested first, then in the parent directory, and then in the directory of the file that is being requested. This is mostly because it is common to have an entry file (often `index.php`) in a `public_html` or `www` directory, with other project files (including for example the `vendor/` directory) in a parent directory. It allows for company/project wide path names as well.

Mappings in files that are scanned later, override already existing mappings.

More specific rules apply before more generic rules.

Xdebug native path mappings are used in the following situations:

1. initiating a debugging connection through the fileurl element of the debug initiation request.
2. setting a breakpoint
3. checking for breakpoints
4. rendering file names where xdebug.file_link_format applies

An application can additionally configure path maps by calling `xdebug_set_source_map()` with a file name as argument. Rules in the parsed file(s) override earlier path mappings of the same type.

The file scanning based approach is necessary so that the fileurl element in the debug initiation request can be mapped, as when an application gets to call `xdebug_set_source_map()` during script execution, it is already too late to apply that mapping to the debug initiation.

File Format

Each mapping file must end in `.map`, and the following stanzas and syntax is supported:

`remote_prefix: <path>` (optional)

Defines the prefix that is prepended to every remote path. Can be present more than once, and applies from the moment it is parsed only.

`local_prefix: <path>` (optional)

Defines the prefix that is prepended to every local path. Can be present more than once, and applies from the moment it is parsed only.

`<remote_path> = <local_path>`

Creates a mapping from `remote_path` to `local_path`. Both of these paths are either:

- a directory name (`cache/`) — **must** end in a `/`.
- a file name (`cache/MyRewrittenClass.php`)
- a file name + line number (range) (`templates/layout.ezt:4-20` or `templates/layout.ezt:24`). A single number means the start and end "range" is the same.

A **remote** path is the directory/file name as seen by Xdebug on the system where it runs, and a **local** path is the directory/file name that an IDE user would edit directly.

Lines starting with `#` are comments.

Empty lines are allowed, and are ignored.

Example Files

Full Directories

This is an equivalent to what IDE's currently provide.

```
/var/www/ = /home/derick/projects/example.com/
```

This one line file maps all files under the `/var/www/` directory to the equivalent file name in the `/home/derick/projects/example.com/` directory.

Alternatives to this mapping file, with the same exact meaning:

```
local_prefix: /home/derick/projects

/var/www/ = example.com/
```

The `local_prefix` gets prepended to `example.com/`, making it `/home/derick/projects/example.com/` again.

File to File Maps

These are primarily for the Neos/Flow use case, that require mapping of file names to other file names.

```
# Path prefixes
remote_prefix: /home/derick/dev/neos/Data/Temporary/Development/Cache/Code/Flow_Object_Classes
local_prefix: /home/derick/dev/neos/Packages

# The mapping table
Neos_Behat_Command_BehatCommandController.php = Application/Neos.Behat/Classes/Command/BehatCommandController.php
Neos_CliSetup_Command_SetupCommandController.php = Application/Neos.CliSetup/Classes/Command/SetupCommandController.php
Neos_CliSetup_Command_WelcomeCommandController.php = Application/Neos.CliSetup/Classes/Command/WelcomeCommandController.php
```

You can see why it makes sense to have the remote_prefix and local_prefix stanzas, as they drastically reduce the contents of the files, also making them more legible.

Line Range Maps

For use with for example templating systems.

```
# Path prefixes
remote_prefix: /home/derick/dev/xdebug.cloud/src/cache/compiled_templates/xhtml-updqr0
local_prefix: /home/derick/dev/xdebug.cloud/src/templates

# The mapping table
user-info-823edfe12e38a649355c5172b9d98e0a.php:2-31 = user-info.ezt:1
user-info-823edfe12e38a649355c5172b9d98e0a.php:32-33 = user-info.ezt:2
user-info-823edfe12e38a649355c5172b9d98e0a.php:34-36 = user-info.ezt:3
user-info-823edfe12e38a649355c5172b9d98e0a.php:37 = user-info.ezt:4
user-info-823edfe12e38a649355c5172b9d98e0a.php:38 = user-info.ezt:5
user-info-823edfe12e38a649355c5172b9d98e0a.php:39-40 = user-info.ezt:6
user-info-823edfe12e38a649355c5172b9d98e0a.php:41-46 = user-info.ezt:7
…
user-info-823edfe12e38a649355c5172b9d98e0a.php:301-302 = user-info.ezt:73
user-info-823edfe12e38a649355c5172b9d98e0a.php:303 = user-info.ezt:74-75
user-info-823edfe12e38a649355c5172b9d98e0a.php:304-306 = user-info.ezt:76
```

As you can see, these files will get pretty big fast.

Tasks and Estimation

- Implement robust file parser with new test framework [16 hours]
- Modify DBGp's breakpoint_set to use source maps [8 hours]

- Update breakpoint_get and breakpoint_list to use source maps, if appropriate [4 hours]
- Modify DBGp's fileuri element to use source maps [2 hours]
- Change breakpoint checking algorithm to use source maps [16 hours]
- Rewrite breakpoints when xdebug_set_source_map() is called [8 hours]
- Investigate all other locations where path names are displayed and used [6 hours]
- Testing many scenarios, integrations, and real life use cases [20 hours]

Total: 80 hours @ £150/hour → £12 000

Future Scope

- A more condensed approach to line range maps
- Extending line range maps to also include column information, although IDEs can't do anything with this yet, and neither does the DBGp protocol support this yet either.
- Functions to invoke the mapper from userland through PHP functions.
- Support source maps when generating profile information
- Support source maps in code coverage information