Debugging

* The act of identifying and removing errors
* PHP provides debugging mechanisms

PHP is an interpreted language

* Scripts are indirectly executed
* Advantages:
* Program flexibility
* Smaller program size
* Dynamic typing
* Disadvantages:
* Slower
* Some errors can’t be detected until execution

Kinds of errors

* Syntax error
* Violation of possible character combination rules
* Execution halted; results in a crash
* Logical/Semantic error
* Operates incorrectly, but within rules; no crash

PHP problem hierarchy

* Provides behavioral expectations to developers
* Error reporting levels
* Severity and consequence
* Timing (When the problem is occurred)

PHP problem severities

* Notice – non-fatal, probably needs improvement
* Warning – non-fatal, but something is wrong and will probably cause errors
* Error – fatal, and the program is terminated (crash)
* Some refers as crash is “white screen of death”

When PHP problems occur

* Compile time
* When script is interpreted (no execution)
* Run time
* During execution (script interpreted correctly)

Error levels

* 16 error-reporting levels
* Each level has a constant name and integer value
* Constant name – used by developers
* Integer value – used internally as bit field

Common error levels

* E\_ERROR – fatal run-time error (crash)
* E.g. running out of memory
* E\_WARNING – run-time warning; does not halt
* E\_PARSE – compile-time error; does not execute
* E\_NOTICE – run-time notice; does not halt, but there’s probably something wrong
* E\_STRICT – compile-time notice; does not halt, but reinforces forward compatibility and best practices
* E\_DEPRECATED – run-time notice; does not halt, but will not work in future versions.
* E\_ALL – combines all errors, warnings, and notices

Configuration: errors and displaying errors

* Error\_reporitng – sets level

1. Set with constant (a bit field)
2. Default: all except E\_NOTICE and E\_STRICT

* display\_errors – displays errors to users

1. 1(on) or 0 (off)
2. Default: 1(on)

Display\_errors 1 only in development server and never shown to users.

Error\_reporting and display\_errors are set in

* Php.ini – PHP configuration, server-wide (which required server restart and affects the entire server)
* Httpd.conf – server configuration (and also required server restarting)
* Htaccess – by directory
* Run time – in the script, always available

Not all configuration options can be changed

Editing php.ini

* Sets configuration for every site on the web server
* Optimal for development
* Location:

1. Different across platforms
2. Generated by MAMP (check docs)
3. Convenient way for identifying php.ini file we use phpinfo();

Error logging configuration

* log\_errors – toggles error logging

1. default: off(0)

* log\_errors\_max\_len – length of error message in bytes

1. default: 1024 (bytes); 0 is unlimited

* error\_log – path and filename of log file

1. default: NULL (Apache error log)

error logging best practice

* do not log PHP errors in Apache error logs
* set log file location out of web root

1. example: /var/log/apache2/php\_errors.log

* for demonstration purpose, placing in local folder Debugging\_in\_PHP
* defining a maximum line length reduces file size

Set

* log\_errors = on
* log\_errors\_max\_len = 0
* Error log file location

error\_log="C:\xampp\php\logs\php\_error\_log"

* For example explicitly set error\_log file location

/media/sf\_sandbox/php\_errors.log

E\_WARNING

* A non-fatal problem
* ini\_set may not able on some third party host
* disable\_functions= ini\_set

is used for disabled a functions for security reasons

trigger\_error

* useful for creating errors for logging
* trigger\_error — Generates a user-level error/warning/notice message
* The trigger\_error() function creates a user-level error message.
* trigger\_error(errormsg,errortype);
* Example

<?php

if ($divisor == 0) {

trigger\_error("Cannot divide by zero", E\_USER\_ERROR);

}

?>

* see <http://www.php.net/manual/en/errorfunc.constants.php>
* E\_USER\_NOTICE (default for trigger\_error)
* E\_USER\_WARNING
* E\_USER\_ERROR

Graceful fatal error handling

* Fatal errors will happen
* Execute function just before shutdown with register\_shutdown\_function
* Shutdown has already started
* error\_get\_last will still work for file, line, type, message
* won’t work if file with register contains a parse error
* register\_shutdown\_function — Register a function for execution on shutdown

error\_get\_last() Function

* The error\_get\_last() function returns the last error that occurred (as an associative array).

The associative array contains four keys:

* [type] - Describes the error type
* [message] - Describes the error message
* [file] - Describes the file where the error occurred
* [line] - Describes the line where the error occurred