1

What's New in PHP5?

So what's the big deal about PHP5? If you're experienced with PHP4, you probably know about object-oriented programming and the way this was handled with PHP4. If you're unfamiliar with PHP, but you're familiar with other programming languages, you'll probably find PHP5's implementation of object-oriented principles familiar. Luckily, things have become a lot easier with the release of PHP5. However, there are other improvements and changes, such as more configuration options in php.ini and a host of new array-related and other functions, besides just "better object-oriented programming" handling. This chapter outlines these changes for you.

Object-Oriented Changes

The changes that follow relate to the OOP model and associated features and related topics. The majority of these changes are covered in greater detail in Chapter 2, but are also briefly outlined here for your quick reference.

Passing Objects

One big impact of OOP changes in PHP5 is the way that variables are passed as parameters to functions. In PHP4, by default, variables were passed by value instead of by reference, unless denoted otherwise with the syntax `&$varname`. In PHP5, the default is to assign a value by reference.

Exceptions

In a nutshell, exceptions are the procedures that happen when something goes wrong. Instead of your program completely halting when it reaches an unexpected error, you can now exert a little more control over what the program should do when it reaches said error. You are probably familiar with the `set_error_handler()` function available in PHP4. If you aren't, the purpose of this function is to define a user function for error handling. However, it had many limitations in its implementation. For example, it would not work if the error was type `E_ERROR`, `E_PARSE`,

E_CORE_ERROR, E_CORE_WARNING, E_COMPILE_ERROR, E_COMPILE_WARNING, and most of E_STRICT. Also, if the error occurred before the set_error_handler() function script, the function would never be called. With PHP5 comes a new framework for handling exceptions.

Try/Catch/Throw

If you have previous programming experience with languages like C++ or Java, you have undoubtedly heard the terms "try," "catch," and "throw." An exception is "thrown" when an error occurs. Code that could possibly cause an error is put in a try/catch block where it attempts to run (under try), and if an exception is thrown, it is "caught" and a user-defined action should be taken (under catch). The syntax for the try/catch/throw block is as follows:

```
<?php

try
{
  $error_message = 'Hello, I am an error.';
  throw new Exception($error_message);
}
catch (Exception $e)
{
  echo 'Exception caught: ',  $e->getMessage(), "\n";
}

?>
```

Built-In Exception Class

The class Exception that you just saw is also an improvement for PHP5, and it is built-in by default. With PHP5, you can now obtain information about exceptions that have been thrown and use that data to react appropriately. The built-in exception class, which processes and stores that data, looks like this:

```
<?php

class Exception
{
   protected $message = 'Exception Thrown';
   protected $code = 0;
   protected $file;
   protected $line;

   function __construct($message = null, $code = 0);

   final function getMessage();
   final function getCode();
   final function getFile();
   final function getLine();
   final function getTrace();
   final function getTraceAsString();

   function __toString();
}

?>
```

Take a closer look at the `Exception` class:

Member or Method Name	What It Represents
`$message`	The exception message; default can be set to whatever you like
`$code`	The user-defined code
`$file`	The filename where the error occurred
`$line`	The line number where the error occurred
`getMessage()`	Returns `$message`
`getCode()`	Returns `$code`
`getFile()`	Returns `$file`
`getLine()`	Returns `$line`
`getTrace()`	Returns an array that contains the information from a `debug_backtrace()` function
`getTraceasString()`	Returns the same result as the `getTrace()` function, but in a formatted string
`toString()`	A PHP5 magic method that allows the object to be formatted to a string when used in conjunction with `print` or `echo`

In the previous `try/catch/throw` example, we used the `getMessage()` function in the built-in exception class to return and display what the error message was.

Extending the Built-In Exception Class

Obviously the built-in exception class is the most generic form of the class, and while it is very useful in a lot of situations, sometimes you may want to be more specific in the ways that your exceptions are handled. For instance, when sending email, you do not receive a PHP error if the recipient's email address is null. Unless you are monitoring your `sendmail` logs, you probably wouldn't know that your email hadn't been sent. This would be helpful information to have, so you can easily extend the built-in class with the `extends` keyword, as follows:

```
<?php

class emailException extends Exception
{
  function __construct($message)
  {
    echo "There was an error sending your email: <br \>";
    echo $message;
  }
}

function sendEmail($to, $subject, $message)
{
  if ($to == NULL)
```

```
    {
       throw new emailException ("Recipient email address is NULL");
    }

    mail($to, $subject, $message);
}

try
{
    sendEmail("", "Exception Testing",
              "We are testing the exception handling in PHP5");
}
catch (emailException $e)
{
    echo " in file <strong>" . $e->getFile() . "</strong>";
    echo " on line <strong>" . $e->getLine() . "</strong>";
}

?>
```

This results in the following:

```
There was an error sending your email:
Recipient email address is NULL in file /usersites/email_exception.php on line 11
```

You were able to access the filename and line information that was stored in the built-in exception class, but you were also able to add a more detailed error message for debugging purposes.

The set_exception_handler() Function

By default, any exception that isn't caught is going to produce a fatal error and halt execution of your script. PHP5 gives you a chance to clean up any uncaught exceptions with the `set_exception_handler()` function. The class name, error message, and a backtrace are passed to the `set_exception_handler` function, but you can also access the built-in exception class information. The function is used as follows:

```
<?php

function lastResort ($e)
{
    echo "Previously uncaught exception being caught from " . get_class($e);
    echo " in file: " . $e->getFile() . " on line " . $e->getLine();
}

set_exception_handler("lastResort");

throw new Exception();

?>
```

Your result will be something like this:

```
Previously uncaught exception being caught from Exception in file:
/usersites/set_exception_handler.php on line 9
```

Note that you must use quotation marks around your function name within the `set_exception_handler()` function; otherwise you will get a notice that you are attempting to use an undefined constant.

Interfaces

Interfaces have been added to PHP5, allowing you to bind classes to more than one place. They consist of empty functions (which will be defined in implementing classes) and any constants you want to apply to the implementing classes. All implementing classes of an interface must provide an implementation of each method defined in the interface, thereby ensuring that future users of your code know what methods to expect.

Interfaces are discussed in greater detail in Chapter 2.

Iterators

Through the Standard Public Library (SPL) you have access to a whole new set of classes and interfaces that allow you to manipulate and utilize iterators. The SPL is an extension that is compiled and available by default in PHP, and it is basically a library of interfaces and classes for you to use. The next section discusses the `Iterator` class itself. You will note that many of the functions are similar to the array functions.

Main Iterator Interface

The `Iterator` class is a built-in interface that allows you to loop through anything that can be looped with a `foreach()` statement — for example, a directory of files, a result from a query, or an array of data. It comes with certain inherent functions:

- `Iterator::current()`: Returns the current element.
- `Iterator::key()`: Returns the current element's key.
- `Iterator::next()`: Moves the pointer to the next element.
- `Iterator::valid()`: Verifies if there is an element present after calling next() or `rewind()`.

By pre-defining the generic Iterator functionality, these can all help you use iteration to make your code more powerful and more efficient. The most important thing about the Iterator interface is that it is one of the building blocks for the other Iterator classes.

Other Iterator Classes

The following sections give a brief description of some of the major classes that inherit from the `Iterator` class. More detailed information about any of the following classes, and a more comprehensive list can be found at the online SPL documentation:
`http://www.php.net/~helly/php/ext/spl/main.html`.

DirectoryIterator

As the name suggests, `DirectoryIterator` is an Iterator class that implements `Iterator` and allows you to work with a directory of files. Within this class lies a powerful set of functions that can glean information about each file such as times and dates of modification, file size, file type, owner of the file, full path of the file, and what permissions are associated with the file (just to name a few).

RecursiveIterator

The `RecursiveIterator` class assists with recursive iteration, or the act of functions being automatically called over and over again until a certain criterion is reached. Functions that come with this class help you determine if there are child iterators, and what those children might be.

ArrayIterator

With the `ArrayIterator` built-in class, you can modify array values and keys while iterating over an object. It allows you to manipulate arrays with each iteration. Functions such as `append()`, `copy()`, and `seek()` enable the coder to work with arrays above and beyond the usual set of array functions.

Iterators are continually being improved upon and enhanced, and you can expect to see some big changes with PHP 5.1.

Constructors and Destructors

In PHP4, you could call a method immediately in a class by naming it the same as the class name. In PHP5, there is a new "magic method" name for this purpose, called `__construct()`. By including this method in your class, it will be called automatically when the object is created.

Likewise, the object will be destroyed when the magic method `__destruct()` is included in your class. This method destroys the object when called. The `__destruct()` method will also be called if there are no more references to your object, or if you reach the end of your PHP script.

Chapter 2 includes a more in-depth discussion of constructors and destructors.

Access Modifiers

You can now control the level of visibility of your class members and methods with three keywords new to PHP5: `public`, `private`, and `protected`. Methods and members are denoted as public, private, or protected.

The following is a brief summary of the `private`, `protected`, and `public` keywords (you can read more about these properties in Chapter 2):

- `Public`: These members and methods are available to the entire script. They can be referenced from within the object or outside of the object.
- `Protected`: Members or methods that are identified as protected are available only from within the object or an inheriting class.
- `Private`: Private methods and members are available only from within the object itself and are not available to any inheriting class.

The final Keyword

The `final` keyword can be used with a class or a method within a class. When used with a class, this keyword blocks any other class from inheriting it. When used with a method, it keeps any inheriting class from overriding the method, effectively ensuring their permanency, and protecting them from other programmers or even yourself.

The static Keyword

Declaring a member or method as `static` binds it to the class under which it resides, and not to any one object or instance of the class. Static members and methods can be accessed throughout the script.

There are some important things to note about the use of the `static` keyword. First, instead of accessing the variable from within the class with the `$this->varName` syntax, you should use `self::$varName`. In the previous example, you accessed the `self::$count` member from a different function, but because static members are available to anything inside the class, you still used the `self::` syntax. Second, when accessing a static member from outside the class, you should use the syntax `className::$varName`. Likewise when accessing a static method from anywhere inside the class, you should use `self::methodName()`, and when accessing it from outside the class, you should use `className::methodName()`.

The use of the `static` keyword is discussed in greater detail in Chapter 2.

The abstract Keyword

The `abstract` keyword is used when you have a high-level class that you know will need to be inherited by more specific, lower-level classes. For example, you can use the abstract class and method as follows:

```
<?php

abstract class getToday {
    public $today = getdate();

    abstract function showCalendar();
}

class monthlyCalendar extends getToday {

     function showCalendar() {
     //show this month's calendar
     }
}

class dailyCalendar extends getToday {

     function showCalendar () {
     //show today's daily calendar
     }
}

class yearlyCalendar extends getToday {

     function showCalendar () {
     //show the calendar for this year
     }

}

?>
```

This abstract class basically says "create a calendar, but then you must specify what type in the child class." As you can see, the `showCalendar()` function is intentionally left blank and abstract in the first class because that class should be extended and the proper calendar shown based on the inherited class that was called. The `getToday()` class was incomplete, and was a perfect candidate for abstraction. Abstract methods, like those in the interface, are effectively blank and require further implementation in child classes.

Some rules when using the abstract class include:

- If you have an abstract method, the entire class must be defined as abstract as well.
- You cannot instantiate an abstract class.
- You can define a class as being abstract without having any abstract methods contained in it.
- The inheriting classes that are implementing abstract methods must have the same (or weaker) visibility than their parents.

In Chapter 2, we discuss abstraction in greater detail.

Built-In Method Overloading Functions

Unlike PHP4, where you had to use the `overload()` function to force an overload check, PHP5 natively allows you to overload any method or member reference that is not what is expected, thus effectively giving you control over what happens next. The new magic methods `__get()`, `__set()`, and `__call()` are used in overloading, as follows:

```
<?php

class overLoad {

  function __get($property) {
   //check to see if the property exists within the class
   //and if not, do something we want it to
   //such as show an error. Otherwise return
   //a value.

  }

  function __set($property, $value) {
   //check to see if the property exists within the class
   //and if it does, set it to the value being passed
   //otherwise do something we want it to, such
   //as show an error.

  }

  function __call($method, $array_of_arguments) {
   //check to see if method exists within the class
   //and if not, do something such as defer it to
   //another class containing the method or
   //print an error statement.
```

```
    }

}
?>
```

Chapter 2 discusses overloading in greater detail.

New Functions

Here is a comprehensive list of all the new functions in PHP5, with the exception of those requiring extensions:

- `array_combine()`: Combines two arrays into one and uses one array for values, the other for keys.
- `array_diff_uassoc()`: Determines the differences between two or more arrays with additional key comparison, determined by the named function.
- `array_udiff()`: Determines the differences between two or more arrays by using a named function for data comparison.
- `array_udiff_assoc()`: Determines the differences between two or more arrays with additional key comparison and by using a named function for data comparison.
- `array_udiff_uassoc()`: Determines the differences between two or more arrays with additional key comparison using a named function and by using a named function for data comparison.
- `array_uintersect()`: Determines the intersection between two or more arrays by using a named function for data comparison.
- `array_uintersect_assoc()`: Determines the intersection between two or more arrays with additional key comparison and by using a named function for data comparison.
- `array_uintersect_uassoc()`: Determines the intersection between two or more arrays with additional key comparison using a named function and by using a named function for data comparison.
- `array_walk_recursive()`: Applies a named function recursively to each element of an array.
- `convert_uudecode()`: Decodes a uuencoded string.
- `convert_uuencode()`: Uuencodes a string.
- `curl_copy_handle()`: Copies a cURL handle along with all of its preferences.
- `date_sunrise()`: Returns the time of sunrise based on given latitude, longitude, zenith, and GMT offset.
- `date_sunset()`: Returns the time of sunset based on given latitude, longitude, zenith, and GMT offset.
- `dba_key_split()`: Splits an index in a string representation into an array representation.
- `dbase_get_header_info()`: Returns the column structure information for a dBase database.

- `dbx_fetch_row()`: Fetches rows from a query-result, but will fail if `DBX_RESULT_UNBUFFERED` is not set in the query.
- `fbsql_set_password()`: Changes a named user's password.
- `file_put_contents()`: Equivalent to opening a file, writing to a file, and closing the file.
- `ftp_alloc()`: Sends the `ALLO` command to an FTP server, which sets aside space for an uploaded file.
- `get_declared_interfaces()`: Returns any declared interface in the script.
- `get_headers()`: Returns the headers sent by the server in response to a HTTP request.
- `headers_list()`: Returns a list of response headers sent (or ready to send).
- `http_build_query()`: Builds a URL-encoded query string from the given array.
- `ibase_affected_rows()`: Returns the number of rows that were affected by the previous query (Interbase).
- `ibase_backup()`: Initiates a backup task in the service manager and returns immediately (Interbase).
- `ibase_commit_ret()`: Commits a transaction, and then returns without closing it (Interbase).
- `ibase_db_info()`: Returns information about a database (Interbase).
- `ibase_drop_db()`: Drops a database (Interbase).
- `ibase_errcode()`: Returns an error code (Interbase).
- `ibase_free_event_handler()`: Frees a registered event handler (Interbase).
- `ibase_gen_id()`: Increments a generator and returns the incremented value (Interbase).
- `ibase_maintain_db()`: Executes a maintenance command on the database server (Interbase).
- `ibase_name_result()`: Gives a name to a result set (Interbase).
- `ibase_num_params()`: Returns the number of parameters in a named query (Interbase).
- `ibase_param_info()`: Returns parameter information from a named query (Interbase).
- `ibase_restore()`: Initiates a restore task in the service manager and returns immediately (Interbase).
- `ibase_rollback_ret()`: Rolls back a transaction without closing it (Interbase).
- `ibase_server_info()`: Returns information about a database (Interbase).
- `ibase_service_attach()`: Connects to the service manager (Interbase).
- `ibase_service_detach()`: Disconnects from the service manager (Interbase).
- `ibase_set_event_handler()`: Registers a user-defined function to be called after certain events (Interbase).
- `ibase_wait_event()`: Waits for an event to be posted by the database (Interbase).
- `iconv_mime_decode()`: Decodes a MIME header field.
- `iconv_mime_decode_headers()`: Decodes more than one MIME header field simultaneously.

- `iconv_mime_encode()`: Creates a MIME header field.
- `iconv_strlen()`: Returns the character count of a string.
- `iconv_strpos()`: Finds the position of the first occurrence of a named character within a string.
- `iconv_strrpos()`: Finds the position of the last occurrence of a named character within the specified range of a string.
- `iconv_substr()`: Returns a portion of a string.
- `idate()`: Formats a local time/date as an integer.
- `imagefilter()`: Applies a named filter to an image.
- `image_type_to_extension()`: Returns an image file's extension.
- `imap_getacl()`: Returns the ACL for a given mailbox.
- `ldap_sasl_bind()`: Binds a resource to the LDAP directory using SASL.
- `mb_list_encodings()`: Returns an array of all supported encodings.
- `pcntl_getpriority()`: Returns the priority of a pid.
- `pcntl_wait()`: Pauses the current process until a child process has exited and returns the child process's id.
- `pg_version()`: Returns the client, protocol, and server version in an array.
- `php_strip_whitespace()`: Removes comments, whitespace, and newlines from source code and returns the result.
- `proc_nice()`: Alters the priority of the current process by a given increment.
- `pspell_config_data_dir()`: Sets the location of language data files.
- `pspell_config_dict_dir()`: Sets the location of the main word list.
- `setrawcookie()`: Equivalent to `setcookie()` without a URL encode of the value.
- `snmp_read_mib()`: Reads and parses a MIB file into the active MIB tree.
- `sqlite_fetch_column_types()`: Returns applicable column types from a given table.
- `str_split()`: Splits a string into an array of characters.
- `stream_copy_to_stream()`: Copies data between streams and returns the amount of data copied.
- `stream_get_line()`: Similar to `fgets()` but allows named delimiter.
- `stream_socket_accept()`: Accepts a connection on a socket created from a previous use of the `stream_socket_server()` function.
- `stream_socket_client()`: Opens a stream to an Internet or Unix domain destination.
- `stream_socket_get_name()`: Returns the name of the socket connection.
- `stream_socket_recvfrom()`: Receives data from a socket up to a specified length.
- `stream_socket_sendto()`: Sends data to a specified socket, whether it is connected or not.
- `stream_socket_server()`: Creates a stream on the specified server.

- `strpbrk()`: Searches a string for any of a list of given characters, and returns the remainder of the string from the first instance of success.
- `substr_compare()`: Compares two strings with named offset and optional case sensitivity.
- `time_nanosleep()`: Delays execution of the script for a number of seconds and nanoseconds.

More information about these functions can be found at the source, `http://www.php.net`.

Other Changes to PHP5

Besides the laundry list of OOP changes and the multitude of new functions available, there are also other improvements that have been made in the release of PHP5.

Configuration Changes

In PHP5 there are numerous changes to the `php.ini` configuration file. These are discussed in greater detail in Chapter 5, but here is a list for your reference:

- `mail.force_extra_parameters`
- `register_long_arrays`
- `session.hash_function`
- `session.hash_bits_per_character`
- `zend.ze1_compatibility_mode`

With these new directives, you can exert a little more control over your PHP environment, which gives you a little more freedom in coding.

MySQLi

MySQL, as you well know, fits well with PHP in delivering database-driven dynamic websites. Thus, it is the release of the MySQLi (MySQL improved) extension that makes life easier for everyone.

Configuration Settings

There are several new MySQLi configuration settings available in `php.ini`. Here's a brief description:

- `mysqli.max_links`: Sets the maximum number of MySQL connections per process.
- `mysqli.default_port`: Sets the default TCP/IP port to connect to the MySQL server.
- `mysqli.default_socket`: Sets the default socket name for connecting to the MySQL server.
- `mysqli.default_host`: Sets the default hostname for connecting to the MySQL server.
- `mysqli.default_user`: Sets the default username for connecting to the MySQL server.
- `mysqli.default_pw`: Sets the default password for connecting to the MySQL server.

Built-in Classes and Properties

With the `mysqli` extension comes a new set of built-in classes and properties that you can access in your PHP scripts. These new methods can also be used as functions if your script is procedural in nature as opposed to being object-oriented. To call the functions procedurally, simply use the syntax `mysqli_` before the method name. For example, to create a new connection using OOP, you would type the following:

```
<?php

$connect = new mysqli("server", "user", "pass", "dbname");

//call methods through the mysqli class

$connect->close();

?>
```

To accomplish the same thing using traditional procedural programming, you would type the following:

```
<?php

$connect = mysqli_connect("server", "user", "pass", "dbname");

//call functions using mysqli_ preface

mysqli_close($connect);

?>
```

The following sections describe the type of functions inherent in PHP5.

class mysqli

This class addresses a basic MySQL connection. The constructor of this class creates a new PHP/MySQL connection, and thus a new `mysqli` object. You would use this class to manipulate or retrieve information about the current connection, or perform basic query functions.

The methods available to this class are:

- `autocommit`: Toggles whether or not transactions are automatically committed to the database.
- `change_user`: Switches to another user.
- `character_set_name`: Returns the default character set.
- `close`: Closes a database connection.
- `commit`: Commits a transaction to the database.
- `connect`: Opens a new connection to MySQL database server (also the mysqli constructor).
- `debug`: Uses the Fred Fish debugging library to debug.
- `dump_debug_info`: Uses the Fred Fish debugging library to dump debugging information.
- `get_client_info`: Retrieves information about the client.

- `get_host_info`: Retrieves information about the connection.
- `get_server_info`: Retrieves information about the MySQL server.
- `get_server_version`: Retrieves the current MySQL server version.
- `info`: Returns information about the most recently executed query.
- `init`: Initializes an object prior to calling the `real_connect` function.
- `kill`: Kills a MySQL thread.
- `more_results`: Looks for results from previously called `multi_query`.
- `multi_query`: Executes one or more queries.
- `next_result`: Returns the next result from previously called `multi_query`.
- `options`: Changes or sets connection options for `real_connect` object.
- `ping`: Pings a server connection or reconnects if there is no connection.
- `prepare`: Prepares a single SQL query statement for execution.
- `query`: Executes a query.
- `real_connect`: Connects to the MySQL database server and allows for additional options or parameters to be set.
- `real_escape_string`: Returns an escaped string for valid use in an SQL statement.
- `rollback`: Rolls back the current transaction.
- `select_db`: Selects the named database as active.
- `set_charset`: Sets the character set to be used.
- `ssl_set`: Sets SSL parameters, enabling a secure connection.
- `stat`: Returns information about the system status.
- `stmt_init`: Initializes a statement for use with `mysqli_stmt_prepare`.
- `store_result`: Transfers a resultset from last query.
- `thread_safe`: Returns whether thread safety is given or not.
- `use_result`: Transfers an unbuffered resultset from last query.

The properties available to this class are as follows:

- `affected_rows`: Gets the number of affected rows in a previous MySQL operation.
- `client_info`: Returns the MySQL client version as a string.
- `client_version`: Returns the MySQL client version as an integer.
- `errno`: Returns the error code for the most recent function call.
- `error`: Returns the error string for the most recent function call.
- `field_count`: Returns the number of columns for the most recent query.

- `host_info`: Returns a string representing the type of connection used.
- `info`: Retrieves information about the most recently executed query.
- `insert_id`: Returns the auto generated id used in the last query.
- `protocol_version`: Returns the version of the MySQL protocol used.
- `sqlstate`: Returns a string containing the SQLSTATE error code for the last error.
- `thread_id`: Returns the thread ID for the current connection.
- `warning_count`: Returns the number of warnings generated during execution of the previous SQL statement.

mysqli_stmt

This class addresses a prepared statement, or an SQL statement that is temporarily stored by the MySQL server until it is needed. An example of a prepared statement is `"SELECT * FROM customer WHERE lastname = ?"`. Then, when you are ready to execute the statement, all you need to do is plug in the value for `lastname`. Note that when you are using these methods as functions in procedural programming, you would use a `mysqli_stmt_` preface to the method name (`mysqli_stmt_close`).

The methods available to this class are as follows:

- `bind_param`: Binds variables to a prepared statement.
- `bind_result`: Binds variables to a prepared statement for result storage.
- `close`: Closes a prepared statement.
- `data_seek`: Seeks to an arbitrary row in a statement resultset.
- `execute`: Executes a prepared statement.
- `fetch`: Fetches the result from a prepared statement into bound variables.
- `free_result`: Frees stored result memory for the given statement handle.
- `prepare`: Prepares a SQL query.
- `reset`: Resets a prepared statement.
- `result_metadata`: Retrieves a resultset from a prepared statement for metadata information.
- `send_long_data`: Sends data in chunks.
- `store_result`: Buffers a complete resultset from a prepared statement.

The properties available to this class are as follows:

- `affected_rows`: Returns affected rows from last statement execution.
- `errno`: Returns an error code for the last statement function.
- `errno`: Returns an error message for the last statement function.
- `param_count`: Returns the number of parameters for a given prepared statement.
- `sqlstate`: Returns a string containing the SQLSTATE error code for the last statement function.

mysqli_result

This class represents the resultset obtained from a query against the database. You use this class to manipulate and display query results.

The methods available to this class are:

- `close`: Closes the resultset (named `mysqli_free_result` in procedural programming).
- `data_seek`: Moves internal result pointer.
- `fetch_field`: Retrieves column information from a resultset.
- `fetch_fields`: Retrieves information for all columns from a resulset.
- `fetch_field_direct`: Retrieves column information for specified column.
- `fetch_array`: Retrieves a result row as an associative array, a numeric array, or both.
- `fetch_assoc`: Retrieves a result row as an associative array.
- `fetch_object`: Retrieves a result row as an object.
- `fetch_row`: Retrieves a result row as an enumerated array.
- `field_seek`: Moves result pointer to a specified field offset.

The properties available to this class are as follows:

- `current_field`: Returns the offset of the current field pointer.
- `field_count`: Returns the number of fields in the resultset.
- `lengths`: Returns an array of column lengths.
- `num_rows`: Returns the number of rows in the resultset.

As you can see, the new mysqli class can assist you in writing more efficient code, and give you additional flexibility and control over the MySQL functions available in PHP 4.

XML Support

PHP5 saw an improvement over PHP4's XML libraries. There are several new XML extensions that have been written using libxml2 for improved standardization and maintenance:

- **DOM:** This new set of functions replaces the DOMXML functions from PHP4. They have been reworked to comply with DOM Level 2 Standards put forth by the W3C.
- **XSL:** Formerly known as the XSLT extension, this extension assists in transforming one XML file to another, using the W3C's XSL stylesheet as the standard.
- **SimpleXML:** This set of functions allows you to extract data from an XML file simply and easily. You can then manipulate, display, and compare attributes and elements using the common array `iterator` `foreach()`.

- **SOAP:** The SOAP extension allows you to write SOAP servers and clients, and requires the GNOME XML Library (libxml) to be installed.

Chapter 8 discusses the XML extensions in greater detail.

Tidy Extension

PHP5 now supports the Tidy library, which is available at `http://tidy.sourceforge.net/`. It assists the coder in cleaning up and perfecting HTML code. This extension is available in the PECL library at `http://pecl.php.net/package/tidy`. For a complete list of the Tidy functions and classes available, you can access the PHP manual at `http://us2.php.net/tidy`.

SQLite

Although SQLite was introduced with later versions of PHP4, it has been improved upon for PHP5. SQLite is akin to a mini SQL server. Numerous classes and methods have been built in to PHP5, and it comes bundled with the installation of PHP5. For more information about SQLite, visit the source website: `http://sqlite.org`.

Summary

While the reworking of the OOP model in PHP5 is undoubtedly the biggest and best improvement over PHP4, there are many other areas that have been improved upon to make the PHP coder's life a little easier.

One of the best aspects of PHP is that it is always changing and growing through incremental improvement, as any Open Source language should. The small improvements that make up PHP5 will help you work better with MySQL, help streamline your code, and give you improved access to the strength of XML. The biggest and best improvement, though, is the inclusion of the OOP model. Using OOP instead of procedural programming will literally change the way you think about code. In the next chapter, you'll find out how to get the most out of this new model in PHP.