PHP Extensions

Example 3-1. Extensions in the php.ini file

```
extension=php_bz2.dll
extension=php_curl.dll
;extension=php_fileinfo.dll
extension=php_gd2.dll
;extension=php_gettext.dll
;extension=php_gmp.dll
;extension=php_intl.dll
;extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_ldap.dll
extension=php_mbstring.dll
;extension=php_exif.dll
extension=php_mysql.dll
extension=php_mysqli.dll
;extension=php_oci8.dll
```

> *For a complete list and explanation of PHP extensions, visit:*
>
> *http://php.net/manual/en/extensions.alphabetical.php.*

PHP has a large amount of libraries available with thousands of lines of well tested code. Example 3-1 is a partial copy of the `php.ini` file showing several **extensions** (libraries) that are available to be activated in PHP. Each of these libraries is C code with a PHP wrapper (for better communication with PHP programs). The code is already compiled (notice the `.dll` extensions). A library that exists in the PHP environment can be activated by removing the comment symbol (`;`) in front of the extension statement in the `php.ini` file. Once the INI file has been saved, PHP and Apache must be reloaded (see Chapter 1 for examples on the location of the `php.ini` file and reloading PHP and Apache). This ease of adding libraries is one of the reasons PHP has become so popular. Additional libraries can be "installed" into PHP using several methods. One of the more popular methods, *Pear* (PHP Extension and Application Repository), handles code distribution and maintenance of third-party libraries.

> *The use of Pear and other third-party library installation methods is beyond the scope of this book. However, you can find additional information at the link below.*
>
> *http://pear.php.net/manual/en/about.pear.php.*

In addition, programmers can "install" their own libraries of code (that are not already compiled) directly into an application via the `require` or `require_once` statement. It is common practice in corporations to include code that may be reused many times (such as accessing a database). By providing this (well tested) code in a local library, any changes (such as the movement of the database to another server) can be handled in one location (the local library file), instead of requiring multiple files to be changed. This also reduces code redundancy and increases its reliability.

While local libraries can contain just methods (functions) of code, it is more common that modules (classes) exist in these libraries. This allows programmers to code with the "black box" concept previously mentioned. The three-tier architecture (explained in Chapter 2) is based on this premise. The code classes can be accessed from the library by referencing the library containing the code (via the `require` or `require_once` statement). Once a reference is made, an instance of the class (object) is then created. Once an instance has been created, the program code has access to all the functionality of the object.