```
if ((isset($_POST['username'])) || (isset($_POST['password'])))
{
$userid = $_POST['username'];
$password = $_POST['password'];
if (!(preg_match("/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/", $password)) ||
(!(strlen($userid) >= 8)))
{
        throw new Exception("Invalid Userid and/or Password Format");
}
else
```

This `if` statement uses the PHP function `preg_match` to determine if the format of the password contains one uppercase letter, one lowercase letter, one number, and at least eight characters. Notice that the regular expression format is the same as used with the HTML5 code (the order of the expression has changed, but it still contains the same information). The PHP `strlen` method also checks the user ID to determine that it has eight or more characters.

If either validation does not pass, the program raises an `Exception`. If both pass, the `else` part of the statement executes to encrypt the password and store the information.

```
$password = password_hash($password, PASSWORD_DEFAULT);
$input = file_get_contents($dog_data_xml);
$find = "</users>";
$newupstring = "<user>\n<userid>" . $userid . "</userid>\n<password>" . $password;
$newupstring .= "</password>\n</user>\n</users>";
$find = preg_quote($find,'/');
$output = preg_replace("/^$find(\n|\$)/m","",$input);
$output = $output . $newupstring;
file_put_contents($dog_data_xml,$output);
$login_string = date('mdYhis') . " | New Userid | " . $userid . "\n";
error_log($login_string,3,$user_log_file);
header("Location: e7login.php");
```

Before inserting the password into the XML file, it must be encyrpted (hashed). The PHP method `password_hash` will convert the password to the format shown previously.

> *Programming note—password_hash has many different options and configurations available for the advanced developer. For more information,*
>
> *visit* http://php.net/manual/en/function.password-hash.php.

`file_get_contents` dumps the contents of the XML user ID/password file into `$input`. `preg_quote` will place backslashes next to any special characters in `$find` (such as the backslash contained in `/users`) to keep PHP from trying to interpret those characters as part of a regular expression. `preg_replace` will use the regular expression `/^$find(\n|\$)/m` to search for `</users>` with and without (`\n`) at the end. Since records in files are determined by the newline character, this will ensure that you find `</users>` at either the end of a line in the file or as part of a line in the file. When `</users>` is found in the file (the contents of the file are in `$input`), it is replaced by `""` (empty string). `preg_replace` will also attempt to place backslashes in any string existing in the second parameter (where the `""` is currently located). If the `$newupstring` is placed in that parameter, the encrypted password would be modified by `preg_replace`. This would cause the passwords to not verify, even if the one entered by the user is correct.