Saving Data as a Comma-Separated Values File

The Export-Csv cmdlet makes it easy to export data as a comma-separated values (CSV) file; all you need to do is call Export-Csv followed by the path to the CSV file. For example, this command uses Get-Process to grab information about all the processes running on the computer, then uses Export-Csv to write that data to a file named C:\Scripts\Test.txt:

[crayon-6a9d7870d52f6412908648/]

By default, data is saved in ASCII format. What if you'd prefer to save the data in Unicode format (or maybe UTF7 or UTF8)? No problem; just add the -encoding parameter followed by the desired format:

[crayon-6a9d7870d52fc250267820/]

You might have noticed that the first line in the resulting CSV file lists the .NET object type:

[crayon-6a9d7870d52ff756758008/]

If you'd just as soon not have the .NET object type in your output file then simply include the -notype parameter:

[crayon-6a9d7870d5300939972622/]

Another parameter you might find useful is -force. Suppose Test.txt is a read-only file: that enables people to access, but not change, the contents of the file. That's great, until it comes time to use Export-Csv and update the contents of the file. Here's what happens if you try to export text to a read-only file:

[crayon-6a9d7870d5302016619066/]

That’s where -force comes into play. Suppose you add -force to your export command:

[crayon-6a9d7870d5303506211887/]

In that case, Windows PowerShell will temporarily clear the read-only attribute from the file, update the contents, and then reset the read-only attribute. The end result: the file gets updated, but when Export-Csv is finished the file will still be marked as read-only.