This document was originally written for ACS regular expressions, but it is really generic and applies to all regular expressions.

Early on, some team members formed “best practices” for regular expression syntax, but then didn’t do a very good job communicating that to the remainder team.  My bad.

In short, here are the best practices (with some details after this):

1. First and foremost: Understand all possible variations of messages that can be logged.  If you don’t understand all of the messages that can be logged, then there’s no way to guarantee that you don’t have false positives and/or false negatives.
2. Rather than choosing sequences that require quoting in XML, choose sequences that do not require quoting.  For example: “(?’clientHost’\S+)” is better than “(?&lt;clientHost&gt;\S+)”
3. Be careful that your filter is strict enough to not force needless filtering in the module from the back-reference
4. Use proper filtering (i.e. search for digits if that’s what you expect),
5. Don’t throw in \s\* (or other sequences) if you don’t expect spaces in sequences, or just expect a single space.  This includes leading or trailing .\*’s.  This adds little value but makes the regular expression harder to understand.

That’s the jist of our best practices.  You can stop reading here if you understand everything above.

Now, to add more information on some of these:

On #3: We have a large number of UNIX/Linux systems managed by a small number of ACS systems.  As such, we don’t want to send log lines to the ACS module that will be filtered out by the module anyway – this causes needless overhead on the OpsMgr system.  For example, if you have a log line like:

Oct  7 17:14:31 sles11-cjc sshd[28386]: Invalid user badguy from 192.168.233.132

then you could use a regular expression like “\s\*Invalid\s+user.\*” (this intentionally violates a number of the rules above).  In this case, say we got something like “Invalid user detected in password file” (just as an example).  That would match the regular expression filter specified, but the module would then toss it away (after needless processing) because it wouldn’t match the back-reference filter.  We should keep the regular expression filters and back-reference filters “lock-step” to avoid that.  If I were writing the regular expression for the above, I’d use a filter like: “\s+sshd\[[[:digit:]]+\]: Invalid user \S+ from \S+” (note that you can’t have leading trailing spaces in a regular expression).  The back-reference filter would be something like “SCX\_MANAGEMENTPACK\_ACS\_DATE\s+(?'process'sshd)\[(?'processId'\d+)\]: Invalid user (?'user'\S+) from (?'clientHost'\S+)”.

In this way, you won’t pass log lines that would be filtered out anyway by the back-reference filter (after needless processing).  Each element of the back-reference filter is accounted for in the regular expression.

On #4: If you’re trying to match something like “sshd[28386]”, then use digits rather than \S+.  So, while a filter like “ssh\[\S+\]” would work, it’s better to use something like “ssh\[[[:digit:]]\]” (note that you can’t use “\d+” in a regular expression on UNIX, but you can in a back-reference filter in .net).  This just helps the false positive situation.

On #5: Take the example log line from above:

Oct  7 17:14:31 sles11-cjc sshd[28386]: Invalid user badguy from 192.168.233.132

You could have a regular expression like:

sshd\s\*\[[[:digit:]]+\]\s\*:\s\*Invalid\s+user\s+\S+\s+from\s+\S+

However, a more readable and equivalent regular expression (assuming #1 is satisfied above) is:

                sshd\[[[:digit:]]+\]: Invalid user \S+ from \S+

Both will match the example log line.  But the later is much easier to read (and thus easier to catch errors in) than the former.

That’s it.  The vast majority of our regular expressions to date have followed these principles.  Barring any significant overlooking, I’d like to see all ACS regular expressions follow these guidelines.

/Jeff