POSIX Bracket Expressions

POSIX bracket expressions are a special kind of character classes. POSIX bracket expressions match one character out of a set of characters, just like regular character classes. They use the same syntax with square brackets. A hyphen creates a range, and a caret at the start negates the bracket expression.

One key syntactic difference is that the backslash is NOT a metacharacter in a POSIX bracket expression. So in POSIX, the regular expression [\d] matches a \ or a d. To match a], put it as the first character after the opening [or the negating ^. To match a -, put it right before the closing]. To match a ^, put it before the final literal - or the closing]. Put together, []\d^-] matches], \, d, ^ or -.

The main purpose of the bracket expressions is that they adapt to the user's or application's locale. A locale is a collection of rules and settings that describe language and cultural conventions, like sort order, date format, etc. The POSIX standard also defines these locales.

Generally, only POSIX-compliant regular expression engines have proper and full support for POSIX bracket expressions. Some non-POSIX regex engines support POSIX character classes, but usually don't support collating sequences and character equivalents. Regular expression engines that support Unicode use Unicode properties and scripts to provide functionality similar to POSIX bracket expressions. In Unicode regex engines, shorthand character classes like \w normally match all relevant Unicode characters, alleviating the need to use locales.

Character Classes

Don't confuse the POSIX term "character class" with what is normally called a regular expression character class. [x-z0-9] is an example of what we call a "character class" and POSIX calls a "bracket expression". [:digit:] is a POSIX character class, used inside a bracket expression like [x-z[:digit:]]. These two regular expressions match exactly the same: a single character that is either x, y, z or a digit. The class names must be written all lowercase. POSIX bracket expressions can be negated. [^x-z[:digit:]] matches a single character that is not x, y, z or a digit. A major difference between POSIX bracket expressions and the character classes in other regex flavors is that POSIX bracket expressions treat the backslash as a literal character. This means you can't use backslashes to escape the closing bracket (]), the caret (^) and the hyphen (-). To include a caret, place it anywhere except right after the opening bracket. [x^] matches an x or a caret. You can put the closing bracket right after the opening bracket, or the negating caret. []x] matches a closing bracket or an x. [^]x] matches any character that is not a closing bracket or an x. The hyphen can be included right after the opening bracket, or right before the closing bracket, or right after the negating caret. Both [-x] and [x-] match an x or a hyphen.

Exactly which POSIX character classes are available depends on the POSIX locale. The following are usually supported, often also by regex engines that don't support POSIX itself. I've also indicated equivalent character classes that you can use in ASCII and Unicode regular expressions if the POSIX classes are unavailable. Some classes also have Perl-style shorthand equivalents.
Java does not support POSIX bracket expressions, but does support POSIX character classes using the \p operator. Though the \p syntax is borrowed from the syntax for Unicode properties, the POSIX classes in Java only match ASCII characters as indicated below. The class names are case sensitive. Unlike the POSIX syntax which can only be used inside a bracket expression, Java's \p can be used inside and outside bracket expressions.

POSIX	Description	ASCII	Unicode	Shorthand	Java
[:alnum:]	Alphanumeric characters	[a-zA-Z0-9]	[\p{L&}\p{Nd}]		\p{Alnum}
[:alpha:]	Alphabetic characters	[a-zA-Z]	\p{L&}		\p{Alpha}
[:ascii:]	ASCII characters	[\x00-\x7F]	\p{InBasicLatin}		\p{ASCII}
[:blank:]	Space and tab	[\t]	[\p{Zs}\t]		\p{Blank}
[:cntrl:]	Control characters	[\x00-\x1F\x7F]	\p{Cc}		\p{Cntrl}
[:digit:]	Digits	[0-9]	\p{Nd}	\d	\p{Digit}
[:graph:]	Visible characters (i.e. anything except spaces, control characters, etc.)	[\x21-\x7E]	[^\p{Z}\p{C}]		\p{Graph}
[:lower:]	Lowercase letters	[a-z]	\p{Ll}		\p{Lower}
[:print:]	Visible characters and spaces (i.e. anything except control characters,	[\x20-\x7E]	\P{C}		\p{Print}

	etc.)				
[:punct:]	Punctuation and symbols.	[!"#$%&'()*+,\-./:;<=>?@[\\\]^_`{|}~]	[\p{P}\p{S}]		\p{Punct}
[:space:]	All whitespace characters, including line breaks	[\t\r\n\v\f]	[\p{Z}\t\r\n\v\f]	\s	\p{Space}
[:upper:]	Uppercase letters	[A-Z]	\p{Lu}		\p{Upper}
[:word:]	Word characters (letters, numbers and underscores)	[A-Za-z0-9_]	[\p{L}\p{N}\p{Pc}]	\w	
[:xdigit:]	Hexadecimal digits	[A-Fa-f0-9]	[A-Fa-f0-9]		\p{XDigit}

Collating Sequences

A POSIX locale can have collating sequences to describe how certain characters or groups of characters should be ordered. E.g. in Spanish, ll like in tortilla is treated as one character, and is ordered between l and m in the alphabet. You can use the collating sequence element [.span-ll.] inside a bracket expression to match ll. E.g. the regex torti[[.span-ll.]]a matches tortilla. Notice the double square brackets. One pair for the bracket expression, and one pair for the collating sequence.

I do not know of any regular expression engine that support collating sequences, other than POSIX-compliant engines part of a POSIX-compliant system.

Note that a fully POSIX-compliant regex engine will treat ll as a single character when the locale is set to Spanish. This means that torti[^x]a also matches tortilla. [^x] matches a single character that is not an x, which includes ll in the Spanish POSIX locale.

In any other regular expression engine, or in a POSIX engine not using the Spanish locale, torti[^x]a will match the misspelled word tortila but will not match tortilla, as [^x] cannot match the two characters ll.

Finally, note that not all regex engines claiming to implement POSIX regular expressions actually have full support for collating sequences. Sometimes, these engines use the regular expression syntax defined by POSIX, but don't have full locale support. You may want to try

the above matches to see if the engine you're using does. E.g. Tcl's regexp command supports collating sequences, but Tcl only supports the Unicode locale, which does not define any collating sequences. The result is that in Tcl, a collating sequence specifying a single character will match just that character, and all other collating sequences will result in an error.

Character Equivalents

A POSIX locale can define character equivalents that indicate that certain characters should be considered as identical for sorting. E.g. in French, accents are ignored when ordering words. élève comes before être which comes before événement. é and ê are all the same as e, but l comes before t which comes before v. With the locale set to French, a POSIX-compliant regular expression engine will match e, é, è and ê when you use the collating sequence [=e=] in the bracket expression [[=e=]].
If a character does not have any equivalents, the character equivalence token simply reverts to the character itself. E.g. [[=x=][=z=]] is the same as [xz] in the French locale.
Like collating sequences, POSIX character equivalents are not available in any regex engine that I know of, other than those following the POSIX standard. And those that do may not have the necessary POSIX locale support. Here too Tcl's regexp command supports character equivalents, but Unicode locale, the only one Tcl supports, does not define any character equivalents. This effectively means that [[=x=]] and [x] are exactly the same in Tcl, and will only match x, for any character you may try instead of "x".