Appendix F: Syntax and Format Reference

- Regular Expression Syntax
- Date and Time Patterns
- Number Format

Regular Expression Syntax

Several transformation patterns allow using Regular Expressions to define the textual pattern we want to find in the host screen.

General Examples

Expression	Search for
Just some text	Specific text, find only “Just some text”
.*\.txt	Text files, like “Readme.txt”
Gr[ae]y	Only “Gray” or “Grey”
Colou?r	Only “Color” or “Colour”
\b[1-9][0-9]{2,4}\b	A number between 100 and 99999
\b[A-Z0-9._%-]+@ [A-Z0-9.-]+\.[A-Z]{2,4}\b	Email address, like “help@softwareag.com”

Host Specific Examples

Search for	Use…	Expression
"1 more >" or "2 more >"	To convert the text "1 more >" or "2 more >" etc. to a button that sends PF11	[1-9] more >
"(x-y)", "(January-march)", etc.	To erase any text with this pattern	\(.*-.*\)

General Rules

- [|] separates alternatives.
- Expressions within parentheses are matched as subpattern groups and saved for further use.
- By default, a quantified subpattern matches as many times as possible without causing the rest of the pattern not to match. To change the quantifiers to match the minimum number of times possible, without causing the rest of the pattern not to match, use a [?] right after the quantifier.

Regular Expression Matching

Expression	Matches
{n,m}	At least n but not more than m times
{n,}	At least n times
{n}	Exactly n times
*	0 or more times
+	1 or more times
?	0 or 1 time
.	Everything except \n in a regular expression within parentheses
^	A null token matching the beginning of a string or line (i.e., the position right after a new line or right before the beginning of a string) in a regular expression within parentheses
$	A null token matching the end of a string or line (that is, the position right before a new line or right after the end of a string) in a regular expression within parentheses
\b	Backspace inside a character class ([abcd])
\b	Null token matching a word boundary (\w on one side and \W on the other)
\B	Null token matching a boundary that isn't a word boundary
\A	Only at the beginning of a string
\Z	Only at the end of a string (or before a new line at the end)
\	New line
\r	Carriage return
\t	Tab
\f	Form feed
\d	Digit [0-9]
\D	Non-digit [^0-9]
\w	Word character [0-9a-z_A-Z]
\W	Non-word character [^0-9a-z_A-Z]
\s	A white space character [\t\n\r\f]
\S	A non-white space character [^ \t\n\r\f]
\xnn	The hexadecimal representation of character nn
\cD	The corresponding control character
\nn or \nnn	The octal representation of character nn unless a back reference.

Expression	Matches
\1, \2, \3 ...	Whatever the first, second, third, and so on, parenthesized group matched. This is called a back reference. If there is no corresponding group, the number is interpreted as an octal representation of a character.
\0	The null character. Any other back-slashed character matches itself.
*?	0 or more times
+?	1 or more times
??	0 or 1 time
{n}?	Exactly n times
{n,}?	At least n times
{n,m}?	At least n but not more than m times

Date and Time Patterns

Date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from ’A’ to ’Z’ and from ’a’ to ’z’ are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (’) to avoid interpretation. Quotation marks ("’’") represent a single quote. All other characters are not interpreted; they’re simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from ’A’ to ’Z’ and from ’a’ to ’z’ are reserved):

Letter	Date or Time Component	Examples
G	Era designator	AD
y	Year	1996; 96
M	Month in year	July; Jul; 07
w	Week in year	27
W	Week in month	2
D	Day in year	189
d	Day in month	10
F	Day of week in month	2
E	Day in week	Tuesday; Tue
a	Am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	978

Pattern letters are usually repeated, as their number determines the exact presentation:

- **Text**: For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.

- **Number:** For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it’s needed to separate two adjacent fields.

- **Year:** For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number. For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.

  For parsing with the abbreviated year pattern ("y" or "yy"), SimpleDateFormat must interpret the abbreviated year relative to a century. It does this by adjusting dates to be within 80 years before and 20 years after the time the SimpleDateFormat instance is created. For example, using a pattern of "MM/dd/yy" and a SimpleDateFormat instance created on Jan 1, 1997, the string "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as May 4, 1964. During parsing, only strings consisting of exactly two digits will be parsed into the default century. Any other numeric string, such as a one digit string, a three or more digit string, or a two digit string that isn’t all digits (for example, "-1"), is interpreted literally. So "01/02/3" or "01/02/003" are parsed, using the same pattern, as Jan 2, 3 AD. Likewise, "01/02/-3" is parsed as Jan 2, 4 BC.

- **Month:** If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.

Examples

The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Date and Time Pattern	Result
yyyy-MM-dd	2006-12-31
dd/MMM/yy	31/Dec/06
mmddyy	123106
d.M.yy	31.12.06
"yyyy.MM.dd G ’at’ HH:mm:ss "	2001.07.04 AD at 12:08:56
"EEE, MMM d, ’’yy"	Wed, Jul 4, ’01
"h:mm a"	12:08 PM
"hh ’o’’clock’ a,"	12 o’clock PM,
"K:mm a,"	0:08 PM,
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss"	Wed, 4 Jul 2001 12:08:56
"yyMMddHHmmss"	010704120856

Number Format

The number format expression is used for customizing the string representation of numbers. It can be used for adding grouping and decimal symbols, scientific notation, special prefixes and suffixes (such as currency signs or ‘%’) and much more. The Format Number Expression allows the user to do so by defining a formatting pattern, a string written in a special syntax explained below, according to which the string representation of the number will be generated.

Several usage examples:

Input	Pattern	Resulting string
56	00000	00056
-56	#; (#)	(56)
234.34556	#0.00	234.35
4238476349587	#,##0	4,238,476,349,587
4238476349587	#,###0	4,2384,7634,9587
0.00000034545	0.#E0	3.5E-7
2347	#$	2347$

The format pattern contains a positive and negative subpattern, for example, "#,##0.00;(#,##0.00)". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern is optional; if absent, then the positive subpattern prefixed with the localized minus sign (code>’-’ for most languages) is used as the negative subpattern. That is, "0.00" alone is equivalent to "0.00;-0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits, minimal digits, and other characteristics are all the same as the positive pattern. That means that "#,##0.0#;(#)" produces precisely the same behavior as "#,##0.0#;(#,##0.0#)".

The prefixes, suffixes, and various symbols used for infinity, digits, thousands separators, decimal separators, etc. may be set to arbitrary values, and they will appear properly during formatting. However, care must be taken that the symbols and strings do not conflict, or parsing will be unreliable. For example: the decimal separator and thousands separator should be distinct characters, or parsing will be impossible.

The grouping separator is commonly used for thousands, but in some countries it separates ten-thousands. The grouping size is a constant number of digits between the grouping characters, such as 3 for 100,000,000 or 4 for 1,0000,0000. If you supply a pattern with multiple grouping characters, the interval between the last one and the end of the integer is the one that is used. So "#,##,###,####" == "######,####" == "##,####,####".

Special Pattern Characters

Many characters in a pattern are taken literally; they are matched during parsing and output unchanged during formatting. Special characters, on the other hand, stand for other characters, strings, or classes of characters. They must be quoted, unless noted otherwise, if they are to appear in the prefix or suffix as literals.

Symbol	Location	Localized?	Meaning
0	Number	Yes	Digit
#	Number	Yes	Digit, zero shows as absent
.	Number	Yes	Decimal separator or monetary decimal separator
-	Number	Yes	Minus sign
,	Number	Yes	Grouping separator
E	Number	Yes	Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix.
;	Subpattern boundary	Yes	Separates positive and negative subpatterns
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage
\u2030	Prefix or suffix	Yes	Multiply by 1000 and show as per mille
¤ (\u00A4)	Prefix or suffix	No	Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
’	Prefix or suffix	No	Used to quote special characters in a prefix or suffix, for example, "’#’#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o’’clock".

Number Format in Different Languages

Although the decimal and the grouping separators used in patterns must be ‘.’ and ‘,’ respectively, they might be replaced with different separators during the formatting process, depending on the application language. For example: using the pattern “#,##0.00” to format the number 18734573.07 will yield the string “18,734,573.07” if the application language is English, and “18.734.573,07” in case the application language is Italian. The default minus sign (normally ‘-‘) is also determined according to the application language.

Scientific Notation

Numbers in scientific notation are expressed as the product of a mantissa and a power of ten, for example, 1234 can be expressed as 1.234 x 10^3. The mantissa is often in the range 1.0 <= x < 10.0, but it need not be. In a pattern, the exponent character immediately followed by one or more digit characters indicates scientific notation. Example: "0.###E0" formats the number 1234 as "1.234E3".

- The number of digit characters after the exponent character gives the minimum exponent digit count. There is no maximum. Negative exponents are formatted using the localized minus sign, not the prefix and suffix from the pattern. This allows patterns such as "0.###E0 m/s".
- The minimum and maximum number of integer digits are interpreted together:
    - If the maximum number of integer digits is greater than their minimum number and greater than 1, it forces the exponent to be a multiple of the maximum number of integer digits, and the minimum number of integer digits to be interpreted as 1. The most common use of this is to generate engineering notation, in which the exponent is a multiple of three, e.g., "##0.#####E0". Using this pattern, the number 12345 formats to "12.345E3", and 123456 formats to "123.456E3".
    - Otherwise, the minimum number of integer digits is achieved by adjusting the exponent. Example: 0.00123 formatted with "00.###E0" yields "12.3E-4".
- The number of significant digits in the mantissa is the sum of the minimum integer and maximum fraction digits, and is unaffected by the maximum integer digits. For example, 12345 formatted with "##0.##E0" is "12.3E3". To show all digits, set the significant digits count to zero. The number of significant digits does not affect parsing.
- Exponential patterns may not contain grouping separators.

Rounding

The number format uses half-even rounding for formatting (round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor).

Format Number Formal Syntax

Pattern

PositivePattern

PositivePattern ; NegativePattern

PositivePattern

Prefix_{opt} Number Suffix_{opt}

NegativePattern

Prefix_{opt} Number Suffix_{opt}

Prefix

any Unicode characters except \uFFFE, \uFFFF, and special characters

Suffix

Unicode characters except \uFFFE, \uFFFF, and special characters

Number

Integer Exponent$_{opt}$

Integer . Fraction Exponent$_{opt}$

Integer

MinimumInteger

Integer

, Integer

MinimumInteger

0 MinimumInteger

0 , MinimumInteger

Fraction

MinimumFraction$_{opt}$ OptionalFraction$_{opt}$

OptionalFraction

OptionalFraction$_{opt}$

Exponent

E MinimumExponent

MinimumExponent

0 MinimumExponent$_{opt}$