Apache Nifi Expression Language Cheat Sheet

Reserved Characters

If these characters are present in attribute names they need to be quoted

$ | { } ( ) [ ] , : ; / * '
(space) \t \r \n

Ex. `${'a:attribute name'}`
`${"a:attribute name"}`

Type Conversion

Coerces from one format to another

toString()	${literal(2):toString() :equals('2')}
toNumber()	${literal('2'):toNumber() :equals(2) }
toDecimal()	${filesize:toDecimal()}

Mathematical

plus()	${fileSize:plus(10)}
minus()	${fileSize:minus(10)}
multiply()	${fileSize:multiply(10)}
divide()	${fileSize:divide(10)}
mod()	${fileSize:mod(10)}
toRadix()	${fileSize:toRadix(10)}

Encode/Decode Functions

${message:function()}
Functions: escapeJson, escapeXml, escapeCsv, escapeHtml3, escapeHtml4, unescapeJson, unescapeXml, unescapeCsv, unescapeHtml3, unescapeHtml4, urlEncode, urlDecode, base64Encode, base64Decode

Logic Operators

isNull()	${filename:isNull()}
notNull()	${filename:notNull()}
isEmpty()	${literal('') :isEmpty()}
equals(string)	${filename :equals('value')}
equalsIgnoreCase (string)	${filename :equalsIgnoreCase('v')}
gt(number)	${fileSize:gt(64)}
ge(number)	${fileSize:ge(64)}
lt(number)	${fileSize:lt(64)}
le(number)	${fileSize:le(64)}
and(bool)	${fileSize:gt(1) :and({fileSize:lt(4)})}
or(bool)	${fileSize:lt(1) :or(${fileSize:gt(4)})}
not()	${filename :endsWith('sv'):not()}
ifElse ('true val', 'falseval')	${filename :endsWith('csv') :ifElse('is csv', 'is not csv')}

Date/Time

format is the java SimpleDateFormat

format(format, zone)	${aDate :format('yy/MM/dd','GMT')}
toDate (format,zone)	${literal('99/12/31') :toDate('yy/MM/dd','GMT')}
now()	${now():toNumber()} *milliseconds since epoch*

Text Search

filename:equals('fizz buzz bazz.txt')

startsWith (string)	${filename :startsWith('fizz')}
endsWith (string)	${filename :endsWith('txt')}
Contains (string)	${filename :contains('buzz')}
in(string, string...)	${literal('NO') :in('NO','NOT')}
indexOf(string)	${filename :indexOf('buzz')} == 5
lastIndexOf (string)	${filename :lastIndexOf('z')} == 13
find(regex)	${filename:find('.*zz')}
matches(regex)	${filename :matches('fizz.*txt')}
jsonPath(path)	${theJson :jsonPath('$.attribute')}

Utilities

These subjectless functions provide useful utilities.

ip()	local ip
hostname(bool)	${hostname(true)} *fully qualified hostname*
UUID()	unique generated UUID
nextInt()	system wide counter, not maintained through restart
literal(value)	${literal(2):gt(1)}
getStateValue (key)	${getStateValue('hash')}
thread()	Thread name

https://nifi.rocks

String Manipulation

Examples use filename equal to 'fizz buzz bazz.txt'

toUpper()	${filename:toUpper()}
toLower()	${filename:toLower()}
trim	${literal('abc '):trim()} *'abc'*
substring(start,end)	${filename:substring(0, 3)} *'abc'*
substringBefore(string)	${filename:substringBefore('zz')} *'fi'*
substringBeforeLast (string)	${filename:substringBeforeLast('zz')} *'fizz buzz ba'*
substringAfter(string)	${filename:substringAfter('zz')} *' buzz bazz.txt'*
substringAfterLast (string)	${filename:substringAfterLast('zz')} *'.txt'*
getDelimitedField(index, delimiter, quote char, escape char, strip char)	${filename:getDelimitedField(2, ' ')} buzz
append(string)	${filename:append('.bck')} *'fizz buzz bazz.txt.bck'*
prepend(string)	${filename:prepend('a ')} *'a fizz buzz bazz.txt'*
replace(search,replace)	${ filename:replace(' ','_')} *fizz_buzz_bazz.txt*
replaceFirst(search, replace	${filename:(' ', '_')} *fizz_buzz bazz.txt*
replaceAll(regex, replace)	${filename:replaceAll ('(\w{4})\s', '!$1')} *!fizz!buzzbazz.txt*
replaceNull(replace)	${idonotexist:('abc') :replaceNull('abc')} *'abc'*
replaceEmpty(replace)	${literal(''):replaceEmpty('abc')} 'abc'
length	${filename:length()} *18*

Multiple Attributes

anyAttribute('attr1','attr2'...)	${anyAttribute('bizz','bazz') :contains('value')}
allAttributes ('attr1', 'attr2'...)	${allAttributes('bizz','bazz') :contains('value')}
anyMatchingAttribute(regex)	${anyMatchingAttributes('b.*zz') :contains('value')}
allMatchingAttributes(regex)	${allMatchingAttributes('b.*zz') :contains('value') }
anyDelineatedValue(value, delimiter)	${anyDelineatedValue(${literal('a-b-c')},'-'):contains('a')}
allDelineatedValues(value, delimiter)	${allDelineatedValues(${literal('a-b-c')}, '-'):contains('a)} *false*
join(string)	${allAttributes('attr1','attr2') :join(', ')}
count()	${allMatchingAttributes('b.*zz') :count()} *number of matching*