nexl scripting language

Introduction

nexl expressions allows to perform a wide variety of data manipulations on a JavaScript primitives, arrays and objects hosted by nexl REST server.

For example :

nexl expression	Explanation
${person~X}	resolves person object and produces XML from that object
${fruits#S&,}	resolves fruits array, sorts array, joins all elements with comma

nexl expression definition :

```
${[JS_VAR_NAME][action1][action2]...}
```

Actions perform data manipulation. nexl engine evaluates actions from the left to right order. Every action in chain gets a value calculated by previous action in chain. Some actions in chain can be ignored in some cases. When actions chain is finished the last calculated value is assigned to whole expression.

JS_VAR_NAME is a name of a JavaScript variable in nexl source. If JS_VAR_NAME is not specified first action in chain will get an undefined value.

Action definition :

```
ACTION_ID[ACTION_VALUE]
```

In our first example above for ${person~X} expression, person is name of a JavaScript variable and ~X is an action, where ~ is ACTION_ID (object operation action) and X is ACTION_VALUE (produce XML)

In the second example above for ${fruits#S&,} expression we have a fruits JavaScript variable and 2 actions in chain #S&, where #S action sorts fruits array and &, action joins array elements with comma.

Depending on ACTION_ID, ACTION_VALUE can be a string and/or another another nexl expression (inner expression). For example ${test@${value}} where @ is an ACTION_ID and ${value} is an ACTION_VALUE. ${value} is an expression itself and can contain its own actions nexl expressions can be nested with unlimited depth

Available actions :

ACTION_ID	Explanation
@	Default value. Applies for undefined values. For all other values is ignored. ACTION_VALUE can be a string and/or internal nexl expression(s).
:	Typecast. Performs typecast for primitive values. Also can convert any value to null or undefined. Available ACTION_ID + ACTION_VALUES are :num :str :bool :null :undefined
.	Object property resolution. Resolves property value for objects. Ignored for all other data types. ACTION_VALUE can be a string and/or internal nexl expression(s).
	Object operations. Performs different operations with JavaScript objects. Ignored for all other data types. Available ACTION_ID + ACTION_VALUES are

~	~K (resolves object keys) ~V (resolves object values) ~O (converts to object) ~X (produces XML) ~Y (produces YAML) ~P (produces key-value pairs)
<	Object key reverse resolution. Resolves a key of JavaScript object by its value. Ignored for all non object types. ACTION_VALUE can be a string and/or internal nexl expression(s).
#	Array operations. Performs different operations with JavaScript arrays. All other data types are ignored. Available ACTION_ID + ACTION_VALUES are #S (sorts asc) #s (sorts desc) #U (unique values) #D (duplicate values) #LEN (elements count) #A (converts to array) #F (gets first array element or undefined)
&	Joins JavaScript array elements with ACTION_VALUE. Ignored for all other data types. ACTION_VALUE can be a string and/or internal nexl expression(s).
+	Add/Merge action. Adds elements to existing JavaScript array or merges two JavaScript objects. Ignores for all other data types. For JavaScript arrays ACTION_VALUE can be a string and/or internal nexl expression(s). For JavaScript object ACTION_VALUE must be internal nexl expression which is evaluated to JavaScript object
-	Eliminates items from JavaScript arrays or JavaScript objects. Ignored for all other data types. ACTION_VALUE can be a string and/or internal nexl expression(s).
^	String operations Available ACTION_ID + ACTION_VALUES are ^U (upper case) ^U1 (capitalizes first letter) ^L (lower case) ^T (trims) ^LEN (length)
!	Undefined value operations !E Converts empty strings, empty arrays and empty objects to undefined !U Evaluates the whole expression to undefined if one of the sub expressions is evaluated to undefined
*	Mandatory value validator action. Throws exception for undefined values. You can provide a custom error message as ACTION_VALUE (including inner nexl expressions)
|	Pushes current value to function parameters stack. Doesn't have an ACTION_ID

The following characters are reserved for future actions and must be always escaped in nexl expressions

```
%
>
=
?
```

There are 3 additional actions :

Accessing array elements [â€¦]
Substring [â€¦]
Function call (...)

Actions can be combined in any order without any limit. Actions count is also unlimited.

Please note ACTION_IDS must be escaped if they are used in ACTION_VALUES

Resolving a JavaScript variable value

The ${distanceToMoon} expression resolves a distanceToMoon variable value. This variable can be JavaScript primitive, array or object.

If distanceToMoon variable is not defined expression will be evaluated to undefined.

Empty expressions ${} are also evaluated to undefined

nexl expression(s) can be a part of some string. For example :

```
sites = ['google.com', 'w3schools.com', 'facebook.com'];
urlTemplate = 'http://www.${sites}';
```

The ${urlTemplate} expression is evaluated to following array

```
["http://www.google.com", "http://www.w3schools.com", "http://www.facebook.com"]
```

Default values action @

ACTION_ID is @
ACTION_VALUE can be a string and/or nexl expression(s)
Applies for undefined values

This action replaces undefined values with value you provide in ACTION_VALUE.
Example :

```
${distanceToMoon@384400}
```

If the distanceToMoon variable is not defined nexl engine will apply a '384400' default value. The '384400' value is treated as string by default, but you can cast it to a number in the following way (see more about typecast in next section)

```
${distanceToMoon@384400:num}
```

The ${@384400} expression is always evaluated to '384400' because a ${} expression is evaluated to undefined value and '384400' default value is applied

The following expression is evaluated to empty string :

```
${@}
```

The default value can be a nexl expression itself. For example :

```
${distanceToMoon@${calcDistance}}
```

If the distanceToMoon is undefined nexl engine will apply a default value provided by ${calcDistance} inner expression

If the ${calcDistance} expression is also evaluated to undefined you can provide an additional default value(s) :

```
${distanceToMoon@${calcDistance}@{val1}@${val2}@384400}
```

Default value can be mixed of text and inner nexl expression(s). For example :

```
${@Hello ${world} 2017}
```

The 'Hello ${world} 2017' is a default value which contains inner nexl expression

---

Typecast :

ACTION_ID is :
ACTION_VALUE can be one of the following num, str, bool, null, undefined
Applies for anything (num, str, bool are applied for primitives; null, undefined for anything)

The following type casts are available

ACTION_ID + ACTION_VALUE	Description	Example
:num	Casts string or boolean values to a number. All other data types are ignored. Illegal numbers are casted to undefined	${stringItem:num}
:str	Casts boolean or numeric values to a string. All other data types are ignored.	${boolItem:str}
:bool	Casts numeric or string values to boolean. All other data types are ignored. Zero numbers are casted to false, non zero numbers casted to true. 'true' string values are casted to true, 'false' string values are casted to false. All other string items are casted to undefined	${numericItem:bool}
:null	Any value of any type is converted to null	${array1:null}
:undefined	Any value of any type is converted to undefined	${obj1:undefined}

Typecast examples

nexl expression	Explanation
${@}	Is evaluated to empty string. Empty expression ${} is evaluated to undefined value. Therefore default value action is applied here. In this expression ${@} default value it an empty string

${@319:num}	Casts a '319' string value to numeric
${@:num}	Casts an empty string to numeric (result is undefined)
${@319:bool}	Casts a '319' string value to boolean (result is undefined)
${@319:bool@hello}	Casts a '319' string value to boolean. The result is undefined. Therefore the second default value action is applied. Final result is 'hello' string
${@319:num:bool}	First casts a '319' string value to numeric, then casts a 319 numeric value to boolean (result is true)
${@true:bool}	Casts a 'true' string value to boolean (result is true)
${:null}	Converts an undefined value to null
${obj1:null}	Converts obj1 to null
${obj1:num}	If the obj1 is not a primitive value this casting action will be ignored. :num, :str and :bool actions are applied only for primitives

---

Object property resolution .

ACTION_ID is .
ACTION_VALUE can be a string and/or nexl expression(s)
Applies for objects

This action resolves object properties.
For example the ${person.city} expression resolves a city property of person object where dot is an ACTION_ID and 'city' is ACTION_VALUE.

this and _parent_ are reserved variable names (do not declare variables with those names in your nexl sources)
this points to a current object instance.
parent points to a parent object instance.

Let's consider few examples for the following nexl source code snippet

```
1
2  prop1 = 'address';
3
4  props = ['name', 'age'];
5
6  person = {
7    name: 'Alex',
8    age: 25,
9    address: {
10     country: 'Canada',
11     city: 'Toronto'
12   }
13 };
14
```

nexl expression	Explanation and result
${person.name}	Resolves a name property of person object. The result is 'Alex'
${person.address.city}	First resolves a person object, then resolves an address property of that object and finally resolves a city property from address object. The result is 'Toronto'
${person.${prop1}.country}	First resolves a person object. Then resolves its property provided by ${prop1} internal nexl expression which is 'address'. Finally resolves a country property of that object The result is 'Canada'
${person.${prop2}}	First resolves a person object. Then resolves its property which is provided by ${prop2} internal nexl expression. The prop2 variable is not declared therefore it's evaluated to undefined. nexl engine skips undefined values when performing object property resolution. So the ${prop2} will be skipped and the whole expression will be equaled to a person object The result is person object
${person.${prop2@}}	First resolves a person object. Then resolves its property provided by ${prop2@} internal expression. The prop2 variable is not declared and evaluated to undefined. The ${prop2@} expression has a default value action with is empty string value. Therefore ${prop2@} expression will be evaluated to empty string. person object doesn't have an empty string property. That's why the whole expression will be evaluated to undefined The result is

	undefined
${person.${props}}	First resolves a person object. Then resolves its property provided by ${props} internal expression. The ${props} expression is evaluated to JavaScript array. Therefore nexl engine performs property resolution for each array element The result is ['Alex', 25]

Let's consider examples for _this_ and _parent_ reserved variable names. nexl source code snippet is

```
dirs = {
  ROOT_DIR: '/home/project',
  HOME_DIR: '${_this_.ROOT_DIR}/nexl',

  SUBDIRS: {
    LOGS_DIR: '${_parent_.ROOT_DIR}/LOGS',
    BACKUP_DIR: '${_parent_.HOME_DIR}/BACKUP',
    SUPER_BACKUP_DIR: '${_this_.BACKUP_DIR}/SUPER'
  },

  FILES: {
    SERVER_LOG: '${_parent_.SUBDIRS.LOGS_DIR}/server.log',
    BACKUP_LOG: '${_parent_.SUBDIRS.SUPER_BACKUP_DIR}/backup.log',
    START_SCRIPT: '${_parent_.HOME_DIR}/start.sh'
  }
};
```

nexl expression	Explanation and result
${dirs.ROOT_DIR}	First resolves a dirs object and then resolves its ROOT_DIR property The result is '/home/project'
${dirs.HOME_DIR}	First resolves a dirs object and then resolves its HOME_DIR property. HOME_DIR contains a ${_this_.ROOT_DIR} nexl expression. In this expression _this_ points to dirs object instance. Therefore ${_this_.ROOT_DIR} is equals to a ${dirs.ROOT_DIR} expression The result is '/home/project/nexl'
${dirs.SUBDIRS.BACKUP_DIR}	This expression is evaluated to a string value which contains a ${_parent_.HOME_DIR} sub expression _parent_ points to a dirs object instance, therefore HOME_DIR is resolved as a property of dirs object The result is '/home/project/nexl/BACKUP'
${dirs.SUBDIRS.SUPER_BACKUP_DIR}	This expression is evaluated to a string which contains a ${_this_.BACKUP_DIR} sub expression _this_ points to dirs.SUBDIRS object instance, therefore BACKUP_DIR dir will is resolved as a property of dirs.SUBDIRS object The result is '/home/project/nexl/BACKUP/SUPER'
${dirs.FILES.BACKUP_LOG}	The result is '/home/project/nexl/BACKUP/SUPER/backup.log'
${dirs.SUBDIRS._parent_.FILES._this_.START_SCRIPT}	This examples shows a complex usage of _this_ and _parent_ pointers. The result is '/home/project/nexl/start.sh'

Object operations ~

ACTION_ID is ~
ACTION_VALUE can be one of the following K, V, O, X, Y, P
Applies for objects (except of O which applies for anything)

Object operations action performs different kind of data manipulations with JavaScript objects in nexl expression

ACTION_ID + ACTION_VALUE	Description	Example
~K	Resolves object key set as array. If the current value for this action is not an object action will be ignored. In this example first nexl engine resolves a obj1 JavaScript variable and if it's a JavaScript object, resolves its key set as array	${obj1~K}

~V	Resolves all object values as array (including sub values). If the current value for this action is not an object action will be ignored. In this example first nexl engine resolves a obj1 JavaScript variable and if it's a JavaScript object, resolves all its values as array	${obj1~V}
~O	Converts current value to object if it's not already an object. This action is ignored for objects In this example first nexl engine resolves an item JavaScript variable and if it's not a JavaScript object, converts it to the object in the following way : `{ "item": ITEM_VALUE }` where ITEM_VALUE is a value which was resolved before undefined values are converted to empty objects	${item~O}
~X	Produces an XML from object. This action is ignored for objects In this example first nexl engine resolves a obj1 JavaScript variable and if it's a JavaScript object, produces an XML from that object	${obj1~X}
~Y	Produces a YAML from object. This action is ignored for objects In this example first nexl engine resolves a obj1 JavaScript variable and if it's a JavaScript object, produces a YAML from that object	${obj1~Y}
~P	Produces a key=values pairs from object (property file). This action is ignored for objects In this example first nexl engine resolves a obj1 JavaScript variable and if it's a JavaScript object, produces a key=value pairs from that object	${obj1~P}

Let's consider few examples for the following nexl source code snippet :

```
person = {
    name: 'Alex',
    age: 25,
    country: 'Canada'
};
```

nexl expression	Explanation and result
${person~K}	Resolves key set of person object as array The result is `["name", "age", "country"]`
${person~V}	Resolves values as of person object as array The result is `["Alex", 25, "Canada"]`
${person~O}	This action is ignored because person is already object
${person.name~O}	First nexl engine resolves a person object, then resolves a name property of that object and ~O action converts last result to object The result is `{ "person.name": "Alex" }`

Object key reverse resolution <

ACTION_ID is <
ACTION_VALUE can be a string and/or nexl expression(s)
Applies for objects

Object key reverse resolution is a way to resolve an object keys by value(s)
Let's say we have the following object

```
person = {
    name: 'Alex',
```

```
    age: 25,
    country: 'Canada'
};
```

The ${obj1} expression resolves a obj1 object. If we add an object key reverse resolution action to this expression ${obj1<Alex} it will resolve all object keys which contain 'Alex' string value ('Alex' is an ACTION_VALUE).
The ${obj1<Alex} expression is evaluated to ['name'] array.

Object key reverse resolution action always produces an array. Array can be empty if there is no matching result.

The following expression will be evaluated to empty array

```
${obj1<25}
```

It happens because 25 is treated as string and not as numeric. To provide a numeric constant value use the following internal expression ${@25:num} which casts '25' string value to numeric

The final expression is

```
${obj1<${@25:num}}
```

And it will be evaluated to ['age'] array

Additionally it's possible to provide a multiple values for object key reverse resolution action by using array.
Let's declare the following array in our nexl source :

```

arr1 = [25, 'Canada', true];

```

And the following expression ${obj1<${arr1}} will be evaluated to

```
['age', 'country']
```

Array operations

ACTION_ID is #
ACTION_VALUE can be one of the following S, s, U, D, LEN, A, F
Applies for arrays (except of A which applies for anything)

Array operations perform different kind of data manipulations with JavaScript arrays in nexl expression

ACTION_ID + ACTION_VALUE	Description	Example
#S	Sorts array in ascending order. If the current value for this action is not an array action will be ignored. In this example first nexl engine resolves a arr1 JavaScript variable and if it's a JavaScript array, sorts it in ascending order	${arr1#S}
#s	Same as previous action, but sorts in a descending order	${arr1#s}
#U	Eliminates all duplicate values from array making array values unique. If the current value for this action is not an array action will be ignored. In this example first nexl engine resolves a arr1 JavaScript variable and if it's a JavaScript array, removes all duplicate values from that array	${arr1#U}
#D	Resolves all repeated (duplicate) items in array. If the current value for this action is not an array action will be ignored. In this example first nexl engine resolves a arr1 JavaScript variable and if it's a JavaScript array, resolves duplicate values only in this array	${arr1#D}
#LEN	Calculates array length. If the current value for this action is not an array action will be ignored. In this example first nexl engine resolves a arr1 JavaScript variable and if it's a JavaScript array, calculates his length	${arr1#LEN}
#A	Converts any item to array if it's not already an array. If the current value for this action is an array action will be ignored. For example if the item = 'test', the result will be a ['test']	${item#A}
#F	Resolves the first array element if array has only one element, or undefined value if array is empty or has more than one element. If the current value for this action is not an array action will be ignored. In this example first nexl engine resolves a arr1 JavaScript variable. If it's a JavaScript array and has only one element, resolves that element. But if that array is empty or has more than one element, makes the result undefined	${arr1#F}

Let's consider few examples for the following nexl source code snippet

```

fruits = ['Mango', 'Banana', 'Apple', 'Banana'];
```

```
primitiveVar = 'test';

person = {
    name: 'Alex',
    age: 25,
    country: 'Canada'
};
```

nexl expression	Explanation and result
${fruits}	Just resolves a fruits array as is The result is `["Mango", "Banana", "Apple", "Banana"]`
${fruits#S}	fruits array is sorted in ascending order The result is `["Apple", "Banana", "Banana", "Mango"]`
${fruits#U}	Duplicate values are removed from fruits array, all items in fruits array become unique The result is `["Mango", "Banana", "Apple"]`
${fruits#D}	Not a duplicate values are removed from fruits array The result is `["Banana"]`
${fruits#A}	This action is ignored because fruits is already an array The result is `["Mango", "Banana", "Apple", "Banana"]`
${fruits#F}	fruits array has 4 elements, therefore it is evaluated to undefined value The result is `undefined`
${fruits#U#s}	First eliminates duplicate values and then sorts in descending order The result is `["Mango", "Banana", "Apple"]`
${primitiveVar#S}	This is evaluated to 'test' string because primitiveVar is not a JavaScript array but a string. Therefore array action is ignored
${person~V#LEN}	1) nexl engine resolves a person JavaScript variable which is object 2) Applies a ~V action for that object. This action resolves all object values as array 3) Applies a #LEN action for that array which calculates its length The result is `3`

Joining array elements &

ACTION_ID is &
ACTION_VALUE can be a string and/or nexl expression(s)
Applies for arrays

This action joins all array elements with ACTION_VALUE delimiter.

Let's consider few examples for the following nexl source code snippet

```
fruits = ['Mango', 'Banana', 'Apple', 'Banana'];
delimiter = '*';
```

nexl expression	Explanation and result
${fruits& }	Joins elements of fruits array with comma.

${fruits&,}	The result is Mango,Banana,Apple,Banana
${fruits&${delimiter}}	Joins elements of fruits array with value provided by ${delimiter} expression, i.e. with asterisk character The result is Mango*Banana*Apple*Banana
${fruits&*}	Joins elements of fruits array with asterisk character. The asterisk character is escaped not to be treated as a mandatory value action The result is Mango*Banana*Apple*Banana
${fruits&\n}	Joins elements of fruits array with LF character The result is Mango Banana Apple Banana
${fruits&\t}	Joins elements of fruits array with TAB character The result is Mango Banana Apple Banana
${fruits&\n${delimiter}}	Joins elements of fruits array with LF and asterisk characters The result is Mango *Banana *Apple *Banana

---

Add/Merge action +

ACTION_ID is +
ACTION_VALUE can be a string and/or nexl expression(s)
Applies for JavaScript arrays and objects

By using add/merge action you can add new elements to array or merge two objects.
This action is ignored for all other data types (works only with arrays and objects)

Let's consider few examples regarding to arrays for the following nexl source code snippet

```
fruits = ['Mango', 'Banana', 'Apple'];

vegetables = ['Tomato', 'Cabbage'];

vegetablePrices = {
    Tomato: 15,
    Cucumber: 8
};

greeting = 'Hello';

```

nexl expression	Explanation and result
${fruits+Annona}	First nexl engine resolves a fruits JavaScript array and then adds a 'Annona' string element to the end of fruits array + is ACTION_ID Annona is ACTION_VALUE (string by default) The result is `["Mango", "Banana", "Apple", "Annona"]`
${fruits+Annona+Kiwi+Orange&,}	Annona, Kiwi and Orange are added to fruits array and then all array elements are joined with comma The result is Mango,Banana,Apple,Annona,Kiwi,Orange
${fruits+${vegetables}}	The ${vegetables} inner expression is evaluated to array and it is added to the end of fruits array The result is `["Mango", "Banana", "Apple", "Tomato", "Cabbage"]`
${fruits+${@79:num}}	The 79 numeric value is added to fruits array because the ${@79:num} expression is evaluated to 79 numeric The result is

	`` ["Mango", "Banana", "Apple", 79] ```
${@Annona#A+${fruits}}	1) ${@Annona} expression is evaluated to a 'Annona' string value (see default value section) 2) The ${@Annona#A} expression has a #A action which converts a 'Annona' string value to ['Annona'] array 3) Adding fruits array to the end of ['Annona'] array The result is ``` ["Annona", "Mango", "Banana", "Apple"] ```
${fruits+${vegetablePrices~K}}	The ${vegetablePrices~K} inner expression resolves a key set of vegetablePrices object as array which are adds that array to the end of fruits array The result is ``` ["Mango", "Banana", "Apple", "Tomato", "Cucumber"] ```
${greeting+World}	greeting is not a JavaScript array or object therefore this action is ignored The result is 'Hello'

Merging objects examples

nexl expression	Explanation and result
${obj1+${obj2}}	If obj1 and obj2 are JavaScript objects the obj2 will be merged to a obj1. Otherwise this action will be ignored. nexl engine performs a deep merge for all sub elements of each object
${obj1+test}	If obj1 is a JavaScript object this action will be ignored because 'test' is a primitive string and cannot be merged to an object

Eliminate -

ACTION_ID is -
ACTION_VALUE can a be primitive or array
Applies for arrays and objects

This action eliminates item(s) from JavaScript arrays or objects.
Examples

nexl source and expression	Explanation and result
nexl source ``` fruits = ['Mango', 'Banana', 'Apple']; ``` nexl expression ${fruits-Banana-Apple}	The 'Banana' and 'Apple' string items are eliminated from fruits array The result is ``` ["Mango"] ```
nexl source ``` mixedArray = [25, 'hello', true, 79]; ``` nexl expression ${mixedArray-${@25:num}}	The 25 numeric element is eliminated from mixedArray. ${mixedArray-25} expression will not eliminate a 25 element because it's treated as '25' string. Therefore we need to use an internal expression which supplies us with a numeric value ${@25:num} The result is ``` ["hello", true, 79] ```
nexl source ``` fruits = ['Mango', 'Lemon', 'Banana', 'Apple']; sourFruits = ['Lemon', 'Apple']; ``` nexl expression ${fruits-${sourFruits}}	All items in sourFruits array are eliminated from fruits array The result is ``` ["Mango", "Banana"] ```
nexl source ``` fruits = ['Mango', undefined, 'Apple']; ```	The ${} expression is evaluated to undefined. All undefined items are eliminated from fruits array The result is

nexl expression

`${fruits-${}}`

```
["Mango", "Apple"]
```

nexl source

```
fruits = ['Mango', undefined, null, 'Apple'];
```

nexl expression

`${fruits-${:null}}`

The ${:null} expression is evaluated to null. All null items are eliminated from fruits array

The result is

```
["Mango", undefined, "Apple"]
```

nexl source

```
car = {
    name: 'BMW',
    model: '501',
    price: 999
};
```

nexl expression

`${car-price}`

The price property is eliminated from car object

The result is

```
car = {
    name: 'BMW',
    model: '501'
};
```

String operations ^

ACTION_ID is ^
ACTION_VALUE can be one of the following U, U1, LEN, T
Applies for strings

String operations action allows to perform basic manipulations with JavaScript strings.
Let's consider each operation with examples in the table followed by this nexl source code snippet

```
greeting = 'Hello, World !';

text = 'i can do it';

untrimmedText = '  I can do it   ';
```

ACTION_ID + ACTION_VALUE	Example	Description
^U	`${greeting^U}`	Upper cases a string The ^L action converts a greeting string to uppercase letters. For all other data types this action is ignored The result is `'HELLO, WORLD !'`
^U1	`${text^U1}`	Capitalizes first letter in string The ^U1 action capitalizes a first letter in the text string. For all other data types this action is ignored The result is `'I can do it'`
^L	`${greeting^L}`	Lower cases a string The ^L action converts the greeting string to lowercase letters. For all other data types this action is ignored The result is `'hello, world !'`
^T	`${untrimmedText^T}`	Trims a string The ^T action trims the untrimmedText string. For all other data types this action is ignored The result is `'I can do it'`
^LEN	`${greeting^LEN}`	Calculates string length The ^LEN action calculates a length of the greeting string. For all other data types this action is ignored The result is `14`

Undefined value operations !

ACTION_ID is !
ACTION_VALUE can be one of the following U, E

ACTION_ID + ACTION_VALUE	Purpose	Example	Explanation and result
!E	Converts empty strings, empty arrays and empty objects to undefined	nexl source `emptyStr = '';` `emptyArr = [];` `emptyObj = {};` nexl expressions `${emptyStr!E}` `${emptyArr!E}` `${emptyObj!E}`	For all those 3 expressions result is undefined
!U	Evaluates nexl expression to undefined if one of the sub expressions is evaluated to undefined	nexl source `text = 'My name is ${name}';` nexl expression `${text!U}`	The text JavaScript variable is referencing to a string value which contains a ${name} sub expression. If the ${name} expression is evaluated to undefined value, the ${text!U} expression will be also evaluated to undefined value.

Mandatory value validator *

ACTION_ID is *
ACTION_VALUE can be a string and/or nexl expression(s)
Applies for for undefined values

Mandatory value validator action throws error for undefined values. You can provide a custom error message in ACTION_VALUE including sub expressions

Let's consider few examples for the following nexl source code snippet

```
item = 'test';
fruits = [];
car = {
    name: 'BMW',
    model: '501',
    price: 999
};
```

nexl expression	Explanation and result
${item*}	In this case mandatory value validator is ignored because item JavaScript variable is not equals to undefined
${someUndeclaredVar*}	The someUndeclaredVar JavaScript variable is not declared in nexl source, therefore we will get the following error message : The [${someUndeclaredVar*}] expression cannot be evaluated to undefined (it has a mandatory value validator). Probably you have to provide it as external arg or check why it calculated to undefined
${text*Please provide a text variable}	The text JavaScript variable is not declared in nexl source. We will get a custom error message provided in nexl expression : Please provide a text variable
${fruits*}	In this case mandatory value validator action is ignored because fruits is not equals to undefined
${fruits!E*Fruits array is empty}	1) Resolving fruits array as ${fruits} 2) Applying a !E action. This action converts empty arrays to undefined value. fruits array is empty, so we get an undefined value 3) Mandatory value validator throw error because it got an undefined value Fruits array is empty
${car<${model}!E*car object doesn't contain a /${model@}/ model}	1) Resolves car object as ${car} 2) Performs object key reverse resolution for car object with ${model} value as ${car<${model}} 3) The model JavaScript variable is not declared in nexl source, therefore we get an empty array as a result 4) !E action converts an empty array to undefined value 5) Mandatory value validator throws custom error message because of undefined value 6) Custom error message has internal nexl expression car object doesn't contain a /${model@}/ model} ${model@} is evaluated to empty string The result is car object doesn't contain a // model

${car.color*^L~O}	1) Resolves car object 2) Resolves a color property of car object 3) The color property is not a member of car object therefore we get an undefined value 4) Mandatory value validator throws error because of undefined value If the color property were a part of car object, mandatory value validator would be skipped and all actions after it would performed

---

Accessing array elements [...]

ACTION_ID is [which ends with]
ACTION_VALUE is located between [...] square brackets
Applies for arrays (and strings)

Array elements can be accessed by their indexes. Array indexes can be :

1) integer number where negative integers are counts from the end
2) ^ character which points to the first element
3) $ character which points to the last element
4) nexl expression which can be evaluated to a 1), 2), 3)

It's possible to specify indexes range.
Also it's possible to specify a multiple indexes/ranges in a single brackets pair.

Let's consider examples for the following nexl source code snippet

```
1
2  fruits = ['Mango', 'Banana', 'Orange', 'Annona', 'Grape'];
3
4  item = 3;
5  item1 = -1;
6  item2 = '$';
7  text = 'hello';
8
9  obj = {};
10
```

nexl expression	Explanation and result
${fruits[0]} ${fruits[^]}	Resolves first array element The result is 'Mango'
${fruits[$]}	Resolves last array element The result is 'Grape'
${fruits[-1]}	Resolves second array element from the end The result is 'Annona'
${fruits[0..1]} ${fruits[^..1]}	Resolves array elements range from 0 to 1 The result is `["Mango", "Banana"]`
${fruits[^..-2]}	Resolves array elements range from first to third from the end The result is `["Mango", "Banana", "Orange"]`
${fruits[1..-1]}	Resolves array elements range from the second to second element from the end The result is `["Banana", "Orange", "Annona"]`
${fruits[-2..$]}	Resolves array elements range from third from the end to the last element The result is `["Orange", "Annona", "Grape"]`
${fruits[0..$]} ${fruits[^..$]}	Resolves array elements range from the first to the last The result is `["Mango", "Banana", "Orange", "Annona", "Grape"]`
	Resolves first, third and last array elements

${fruits[^, 2, $]}	The result is `["Mango", "Orange", "Grape"]`
${fruits[^, 1..2, $]}	Resolves first array element, range from second to third and last element The result is `["Mango", "Banana", "Orange", "Grape"]`
${fruits[${item}]}	Resolves array element with index provided by ${item}. The result is 'Annona'
${fruits[0, ${item1}..${item2}]}	Resolves first array element and range provided by ${item1} and ${index2} indexes The result is `["Mango", "Annona", "Grape"]`
${fruits[${text}]}	Error message. Array index must be an integer or one of the following strings '^', '$'
${obj[3]}	Accessing array elements action is ignored for objects and other non-array data types

---

Substring [...]

ACTION_ID is [which ends with]
ACTION_VALUE is located between [...] square brackets
Applies for strings

All rules are same to array elements access from previous section.

Let's consider few examples for the following nexl source code snippet

```
text = 'Hello, World !';
```

nexl expression	Explanation and result
${text[0]} ${text[^]}	Resolves first character The result is 'H'
${text[5, 13]}	Resolves character number 6 and 14 (count starts from zero). This action produces array of characters The result is `[ ",", "!" ]`
${text[5, 13]&}	Same to a previous example but here we are joining all array element with empty string by using of & action The result is ',!'
${text[^..4, 7..-2]}	Cuts the 'Hello' and 'World' words from the text The result is `["Hello", "World"]`
${text[^..4, 7..-2]& }	Cuts the 'Hello' and 'World' words from the text and joins them with space The result is 'Hello World'

Function call ()

ACTION_ID is (which ends with)
ACTION_VALUE is located between (...) parentheses
Applies for functions

To perform additional data manipulations you can call a JavaScript functions directly from nexl expressions.
For example if you need to perform array sort with special rules :

```
${arraySpecialSort( ARGUMENTS_LIST... )}
```

Where arraySpecialSort() is a JavaScript function.
Here ARGUMENTS_LIST can be only nexl expressions comma delimited as following (spaces are ignored)

```
${arraySpecialSort( ${item1}, ${item2}, ... )}
```

To pass a constant value to the function use nexl expression with default value action :

```
${@hello}
${@79:num}
```

Function call with constant values as arguments :

```
${arraySpecialSort( ${@hello}, ${@79:num} )}
```

Function call examples

nexl expression	Explanation and result
nexl source `function arraySpecialSort(arr1, rules) {` `  // function stuff goes here...` `  return result;` `}` nexl expression `${arraySpecialSort( ${obj1~K}, ${x} )}`	In this example we are calling a arraySpecialSort() function and passing to a function two following arguments : `${obj1~K}` `${x}`
`${Math.round( ${value} )}`	Calling a standard Math.round() function and passing a ${value} argument
`${Math.round( ${Math.PI} )}`	Calling a standard Math.round() function and passing a ${Math.PI} argument. The result is `3`
`${Math.round( ${@3\.14:num} )}`	Calling a standard Math.round() function and passing a 3.14 constant value Let's take a closer look to a ${@3\.14:num} expression. In the ${@3\.14:num} expression we must escape a dot not to be treated as object property resolution action. :num casts a '3.14' string value to numeric The result is `3`

nexl.nexlize() global function gives you ability to use a nexl API from inside your custom JavaScript function.
Function definition :

```
nexl.nexlize(nexlExpression, args)
```

nexlExpression can be a string, array or object which are contain internal nexl expression. nexlize() function will expand those internal expressions

args is optional object to override data elements in nexl source. See more about external arguments here

Example :

```
function callNexlAPI( arg1, arg2 ) {
  // function stuff...
  var result = nexl.nexlize( '${obj1~V}', { ENV: 'dev' } );
  return result;
}
```

System and user functions

Functions declared under nexl.system.functions and nexl.user.functions are global and accessible directly in nexl expressions without prefix.

nexl provides set of useful functions under nexl.system.functions object.
Do not override that object. Declare your custom functions under the nexl.user.functions to extend your code.

For example :

```
nexl.user.functions.increase = function (num) {
  return num++;
};
```

You can call a increase() function in two ways :

```
1) ${increase( ${item} )}
2) ${nexl.user.functions.increase( ${item} )}
```

List of system functions

replaceAll(entity, searchItem, replaceItem)
replaces all searchItem elements in array or all searchItem characters in string with replaceItem

not(param)
inverts boolean

makeObj(key, value)
makes new JavaScript object. If key is not provided the object will be empty

isContains(entity, item)
if entity array contains item or string entity contains item, returns true, otherwise false

isEquals(entity1, entity2)

isBool(item)

isStr(item)

isNum(item)

isNull(item)

isUndefined(item)

isPrimitive(item)

isArray(item)

isObject(item)

ifContains(entity, item, thenIf, elseIf)
if entity array contains item or string entity contains item, returns thenIf, otherwise returns elseIf

ifEquals(entity1, entity2, thenIf, elseIf)

ifBool(item, thenIf, elseIf)

ifStr(item, thenIf, elseIf)

ifNum(item, thenIf, elseIf)

ifNull(item, thenIf, elseIf)

ifUndefined(item, thenIf, elseIf)

ifPrimitive(item, thenIf, elseIf)

ifArray(item, thenIf, elseIf)

ifObject(item, thenIf, elseIf)

Pushing value to function parameters stack |

ACTION_ID is |
ACTION_VALUE - doesn't have a value

Each nexl expression has a special data stack. This data stack can be used when you call a function in nexl expression. In this case all values from that data stack are passed to function arguments (stack arguments are passed prior to arguments described in function call in nexl expression).

You can push values to the stack multiple times.
After each push current chain value is reseted (i.e. became undefined value).
Stack lives within nexl expression. After expression calculation stack is deleted.

Literally stack allows you to pass a value calculated by actions in nexl expression to the function for further calculations

Examples :

nexl expression	Explanation and result
${obj|isObject()}	1) Resolves obj object 2) Pushes it to the stack. This action resets current value in actions chain to undefined 3) Resolves a isObject function. It is a system function therefore it's available globally 4) Calls a isObject() function. This function gets a obj as argument because it was pushed to the stack Expression in this example is same to the following expressoin ${isObject(${obj})}
${item|isPrimitive()|not()}	Resolves item, pushes its value to the stack and then calls a isPrimitive() function. Function evaluation result is also pushed to the stack and then not() function inverts the result
${item1|item2|ifEquals(${a}, ${b})}	1) Resolves item1 and pushes it to stack. 2) Resolves item2 and pushes it to stack. 3) Calls a ifEquals() function. The ifEquals() function accepts 4 arguments. First two arguments are taken from the stack and the followed arguments are taken from explicit function parameters as ${a} and ${b} If item1 and item2 equals the result will be ${a}, otherwise ${b}