

## Documentation guide for nextl-js

[Introduction](#)  
[Products and installation](#)  
[nextl-server](#)  
[nextl expressions](#)  
[nextl-client GUI application](#)  
[nextl expressions are viral](#)  
[Include directive @](#)  
[External args](#)  
[nextl.defaultExpression and nextl.defaultArgs](#)  
[nextl-server additional REST API](#)  
[Compatibility with nextl 1.x version](#)

### Introduction

nextl-js is :

- REST server
- Scripting language

nextl REST server hosts JavaScripts files and exposes their primitives, arrays and objects via REST API. You might need it to store a distributed configuration/data of different servers, clients etc. in a centralized place.

nextl scripting language is intended to perform a wide variety of data manipulations with JavaScript primitives, arrays and objects hosted by nextl REST server. For example you can merge two or more JavaScript objects and produce an XML of them; or to join few arrays, eliminate duplicate elements, sort the rest elements and join them with comma. All those actions can be performed in a single nextl expression while accessing data on a nextl REST server.

In other words you store your configuration/data in native JavaScript form and access it via REST by using nextl expression language which gives you a lot of power to make data manipulations on the fly.

What can nextl scripting language do :

- to resolve object's key set and values
- to resolve object key by value ( object key reverse resolution )
- to produce an XML, YAML and key-value pairs ( property file ) from JavaScript objects
- to sort arrays, to remove/leave duplicate array elements
- to get array elements by indexes ( including negative indexes and special characters )
- to eliminate items from arrays and objects
- to merge objects, to join arrays
- to cast data types
- to perform string operations
- to use nested nextl expressions ( i.e. expression in expression, 'â€¦' )
- to use `_this` and `_parent` special keywords for JavaScript objects
- to use JavaScript functions to perform additional data manipulations including user and system functions
- and moreâ€¦

### Products and installation

nextl-js suggests two following products :

- [nextl-server](#) ( is a nextl REST server )
- [nextl-client](#) ( GUI application to interact with nextl-server and to simulate server work locally )

nextl-js is written in JavaScript and requires [nodejs](#) to run. On the one hand nextl-js depends on nodejs, on the other hand you get a cross platform application.

First download and install a latest version of [nodejs](#). After that use a [command line](#) to install, uninstall and run nextl-js products ( npm program is a part of nodejs installation kit )

Product	Install latest	Run	Uninstall
nextl-server	npm i nextl-server -g	nextl-server	npm uninstall nextl-server -g
nextl-client	npm i nextl-client -g	nextl	npm uninstall nextl-client -g

### nextl-server

nextl-server hosts JavaScript files and exposes their primitives, arrays and objects via REST API. Those files are called nextl sources. By default nextl server is looking up for nextl sources in a `$(HOME)/nextl-sources` directory ( `%userprofile%\nextl-sources` in Windows ).

The following command explains how to change nextl server defaults :

```
nextl-server --help
```

General information about nextl-server

- nextl server accepts GET and POSTS requests ( the result is same )
- all requests are stateless
- nextl-server supports JSONP requests

Let's consider the following example. Run nextl-server, create nextl-sources directory ( `$(HOME)/nextl-sources` ) and put there the following JavaScript file named `example.js` with the following content :

```

1 |
2 | distanceToMoon = 384400;
3 |
4 | fruits = ['Mango', 'Banana', 'Orange', 'Annona', 'Grape'];
5 |
6 | person = {
7 |   name: 'Alex',
8 |   age: 25,
9 |   country: 'Canada'

```

```
10 };
11
```

Here we have 3 items : `distanceToMoon` primitive, `fruits` array and `person` object. Those items are exposed now via REST API. They are accessible by the following URLs :

```
http://localhost:8080/example.js?expression=${distanceToMoon}
http://localhost:8080/example.js?expression=${fruits}
http://localhost:8080/example.js?expression=${person}
```

The URL consists of the following :

URL part	Explanation
localhost:8080	Hostname and port of your next server
/example.js	Relative path to a next-source file. This path is relative to a <code>\$(HOME)/next-sources</code> directory ( <code>%userprofile%\next-sources</code> in Windows ).  You can put as much files as you need in a next-sources directory. Also you can use subdirectories ( path is changed according to a directories structure )  You can't access files in upper level of next-source directory
expression=\${distanceToMoon} expression=\${fruits} expression=\${person}	expression tells us what exactly we need to get from next-source file : <code>distanceToMoon</code> , <code>fruits</code> , <code>person</code> variables.  <code>\$(distanceToMoon)</code> , <code>\$(fruits)</code> , <code>\$(person)</code> are next expressions. See more about next expressions below

## next expressions

next expressions describe what exactly is to get from the next source files and what kind of data manipulations is to perform.

Let's consider examples of next expressions ( below will be explained how to test them )

next expression	Explanation
<code>\$(person.name)</code>	First resolves <code>person</code> object and then resolves a <code>name</code> property of that object
<code>\$(fruits#S)</code>	Resolves a <code>fruits</code> array and sorts it
<code>\$(fruits#S&amp;.)</code>	First sorts <code>fruits</code> array and then joins all elements with comma
<code>\$(fruits[-1])</code>	Resolves a second element from the end of <code>fruits</code> array
<code>\$(person~K)</code>	Resolves a key set of <code>person</code> object as array
<code>\$(person~V&amp;.)</code>	Resolves all values of <code>person</code> object as array and then joins all array elements with comma
<code>\$(person~K#s[\$])</code>	Resolves a key set of <code>person</code> object as array, sorts this array in descending order and finally resolves last array element
<code>\$(person.country[3..\$]^U1)</code>	Resolves a <code>country</code> property of <code>person</code> object, substrings it from fourth element to the end and then capitalizes a first letter
<code>\$(person~X)</code> <code>\$(person~Y)</code> <code>\$(person~P)</code>	Produces an XML, YAML and key-value pairs ( property file ) from <code>person</code> object
<code>\$(person&lt;Alex)</code>	Resolves a key of <code>person</code> object by 'Alex' string value ( i.e. makes object property reverse resolution ). The result is array
<code>\$(person&lt;Alex[0])</code>	Resolves a key of <code>person</code> object by 'Alex' value as array and then resolves a first array element
<code>\$(person~K+\$(person~V)&amp; )</code>	Joins two arrays. The first array is a key set of a <code>person</code> object, the second array are values of a <code>person</code> object. Finally joins all array elements with tab character
<code>\$(distanceToMoon~O)</code>	Converts a <code>distanceToMoon</code> primitive number to JavaScript object
<code>\$(distanceToMoon~O~P)</code>	Converts a <code>distanceToMoon</code> primitive number to JavaScript object and then produces a key-value pair from it
<code>\$(distanceToMoon~O+\$(person))</code>	Converts a <code>distanceToMoon</code> primitive number to JavaScript object and then merges to him <code>person</code> object
<code>\$(Math.PI)</code>	Resolves a <code>PI</code> property from <code>Math</code> object
<code>\$(Math.PI Math.round())</code>	Resolves a <code>PI</code> property from <code>Math</code> object and pushes it to stack. Calls a <code>Math.round()</code> function which automatically gets a <code>Math.PI</code> argument from the stack
<code>\$(Math.PI distanceToMoon Math.max())</code>	Pushes a <code>Math.PI</code> to the stack, then pushes a <code>distanceToMoon</code> to the stack. Finally calls a <code>Math.max()</code> function which gets arguments from the stack

Each next expression from examples above you can test by substituting them to the following URL

```
http://localhost:8080/example.js?expression=${person.name}
```

Please pay attention for HTTP GET requests you have to escape the following characters in the URL & ( %26 ), # ( %23 ), + ( %2B ).

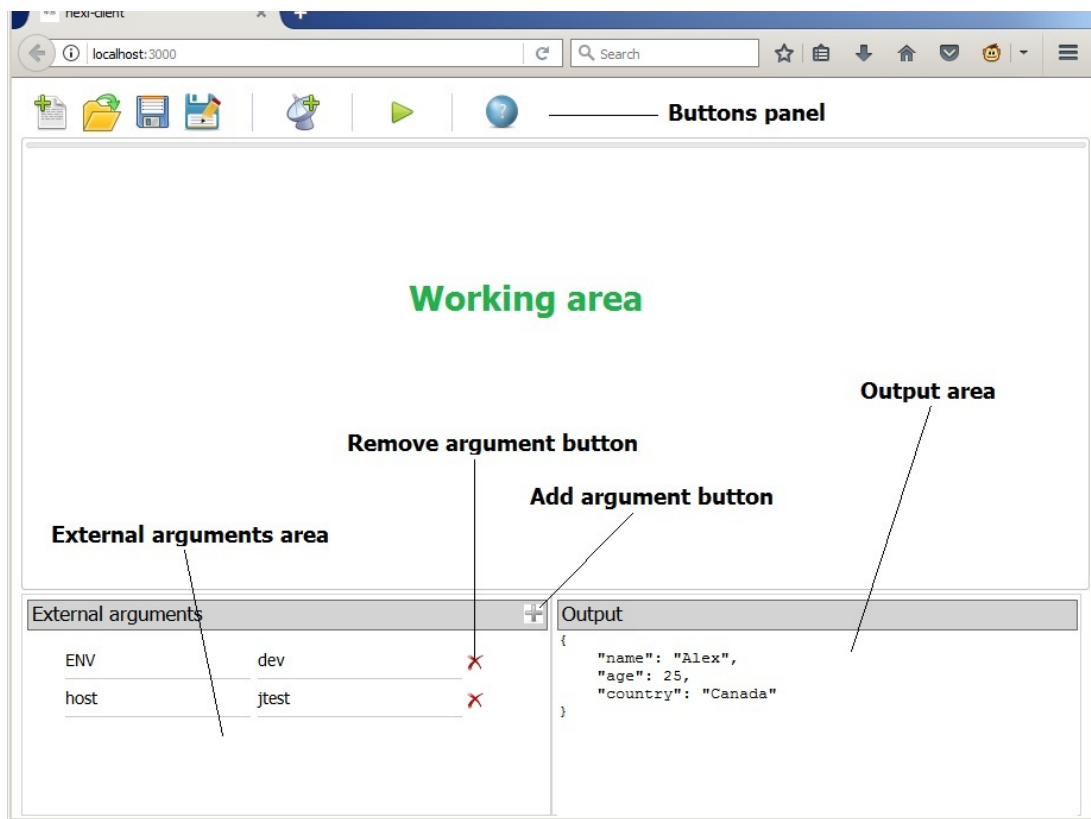
The better way to test/develop next expressions is to use a next GUI client ( it escapes special characters automatically )

Click [here](#) for full spec and more examples of next scripting language

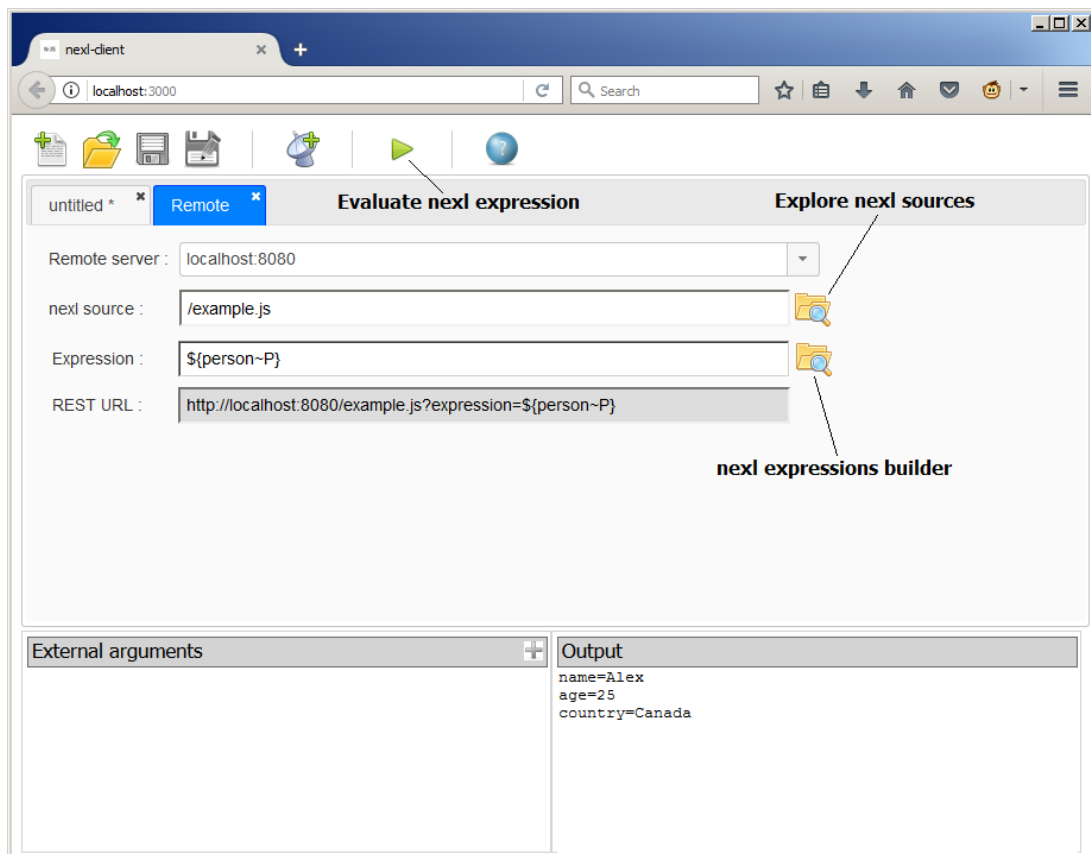
## next-client GUI application

`next-client` GUI application is intended to interact with `next-server` and to simulate its work locally. `next-client` is a Web application and working in a browser. Type `next` in command line to run `next-client`. It will open your default browser with `next-client` application in it ( it's recommended to use Chrome browser )





To interact with remote nexl-server click on "New remote nexl source" button



Remote server is an IP address and port of remote nexl-server

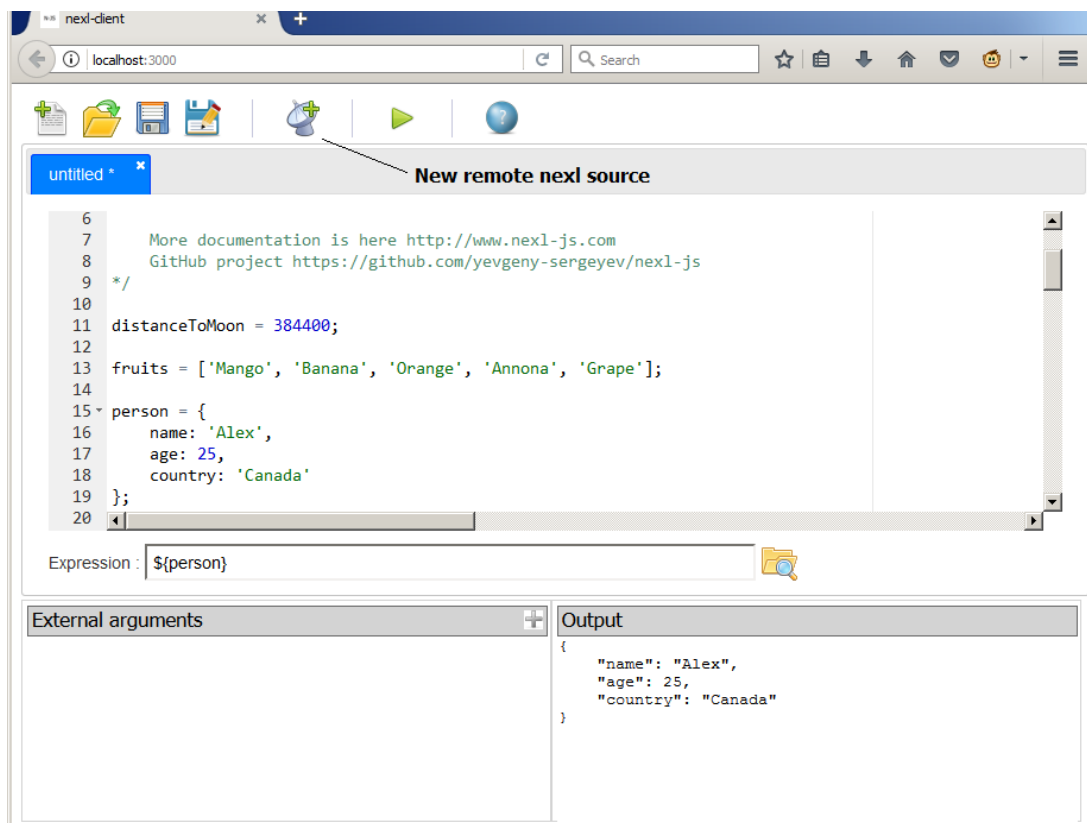
nexl source is a name of JavaScript file hosted and exposed by nexl-server. Specify relative path and file name manually or use "Explore nexl sources" button.

Expression tells us what exactly we need to resolve from the nexl source file. You can specify the expression manually or use "Expression builder" button


REST URL is automatically assembled according to remote server, nexl source, expression and external arguments ( see below ).

If you need to develop you nexl source file without running nexl server, click on "New nexl source file" button  
It will open a JavaScript editor





Here you can edit your nexl source file and test nexl expressions. You can save that file in your local file system. Also you can open existing JavaScript file.

To evaluate nexl expression click on "Evaluate nexl expression" button  ( or press F9 )  
Evaluation result is shown in output area.

nexl-client will be improved in future versions.

### nexl expressions are viral

nexl engine performs deep resolution for all nexl expressions are appeared in JavaScript strings. String items in arrays and objects ( including object keys ) which contains nexl expressions are also evaluated and substituted.

Let's consider the following example :

```

1 |
2 | A = 3;
3 |
4 | B = `${A}`;
5 |
6 | C = `${B}`;
7 |
8 | D = [1, 2, `${C}`];
9 |
10 | E = {
11 |   key1: 'test',
12 |   key2: `${D}`,
13 |   `${A}`: `${C}`,
14 |   key3: 'hello \`${world}`'
15 | };
16 |

```

B is a string variable which contains a `${A}` nexl expression. It will be evaluated to 3.

C is also string variable which is referencing to B variable. Will be evaluated to 3.

D is a JavaScript array where last element is string and contains nexl expression. This array will be evaluated to `D = [1, 2, 3]`;

E is a JavaScript object. All keys and string values which contains nexl expressions will be processed and evaluated by nexl engine.  
The `'hello \`${world}`'` will not be treated as nexl expression because `$` sign is escaped.  
Finally E object will be evaluated to :

```

1 |
2 | E = {
3 |   key1: 'test',
4 |   key2: [1, 2, 3],
5 |   3: 3,
6 |   key3: 'hello ${world}'
7 | };
8 |

```

### Include directive @

nexl sources can include each other by using a `@` directive.  
Let's say you have two JavaScript files : `common.js` and `server.js`  
`server.js` can include a `common.js` file in the following way :

```

1 |
2 | "@ ./commons/common.js";
3 |

```

Path is relative to a path of `server.js` file.  
Path can be absolute.  
Duplicate includings are ignored

---

## External args

JavaScript variables in next source file can be overridden by external arguments when performing a request to next REST server. You can provide a variable name and its value in HTTP request. Variable value can be only a primitive. If you provide a variable which doesn't exist in next source it will be added to next source for that HTTP request only. Variables are provided by external arguments are strings by default. You can cast them to needed type by adding the following to the end of their values :num, :str, :bool, :null, :undefined

Examples :

The following URL just resolves person object

[http://localhost:8080/example.js?expression=\\${person}](http://localhost:8080/example.js?expression=${person})

The following URL

[http://localhost:8080/example.js?expression=\\${person}&person.age=26](http://localhost:8080/example.js?expression=${person}&person.age=26)

is evaluated to the object :

```
1 {
2   {
3     name: 'Emily',
4     age: 26,
5     country: 'Canada'
6   };
7 }
```

By providing a `person.age=26` argument person's age is overridden with '26' string value.

If we need age to be a numeric we need cast a string value to numeric in the following way :

[http://localhost:8080/example.js?expression=\\${person}&person.age=26:num](http://localhost:8080/example.js?expression=${person}&person.age=26:num)

The result is :

```
1 {
2   {
3     name: 'Emily',
4     age: 26,
5     country: 'Canada'
6   };
7 }
```

Please pay attention to the age field. Now it's without quotes because it's become numeric

In the following example we resolve a `${person.${field}}` expression where `${field}` is internal next expression

[http://localhost:8080/example.js?expression=\\${person.\\${field}}&field=name](http://localhost:8080/example.js?expression=${person.${field}}&field=name)

`field` value is provided by `field=name` external argument.

The result is :  
'Alex'

External arguments are also available by the `next.args` reference. You can use it in next expressions and in the functions

---

## next.defaultExpression and next.defaultArgs

When you are querying next REST server you pass the following in the URL :

- next source
- next expression
- optional external args

By using `next.defaultExpression` and `next.defaultArgs` system variables you can embed your next expression and args into next source file in the following way :

```
1 |
2 | next.defaultExpression = "${...}";
3 |
4 | next.defaultArgs = {
5 |   person: {
6 |     name: 'Emily'
7 |   }
8 | };
9 |
```

In that case if you don't provide a next expression and external arguments explicitly in the URL those default values will be applied.  
For example :

<http://localhost:8080/example.js>

`next expression` and `arguments` explicitly provided in HTTP request are always override `next.defaultExpression` and `next.defaultArgs`

---

## next-server additional REST API

next servers has an additional REST API

</next-rest/list-next-sources>

Returns list of all next sources under the `$(HOME)/next-sources` directory (or under a directory you provided as command line argument for next-server)

</next-rest/list-js-variables?nextSource=example.js>

Returns list of all JavaScript variables for next source provided by `nextSource` URL parameter

---

### Compatibility with next 1.x version

next 2.x and 1.x versions are pretty different. It's strongly recommended not to use a 1.x version because of security issues and escaping bugs.

If you have a 1.x version uninstall it in command line :

```
npm uninstall next-server -g  
npm uninstall next-client -g
```

And then install latest version

```
npm i next-server -g  
npm i next-client -g
```

---