NetXMS Scripting Language

http://netxms.org/

Version 4.5, 26 April, 2024

Table of Contents

Introduction

In many parts of the system, fine tuning can be done by using NetXMS built-in scripting language called NXSL (stands for NetXMS Scripting Language). NXSL was designed specifically to be used as embedded scripting language within NetXMS, and because of this has some specific features and limitations. Most notable is very limited access to data outside script boundaries – for example, from NXSL script you cannot access files on server without explicit permission, nor call external programs, nor even access data of the node object other than script is running for without explicit permission. NXSL is interpreted language – scripts first compiled into internal representation (similar to byte code in Java), which than executed inside NXSL VM.

Script security

Because NXSL provides functions for searching objects, and because all scripts are executed on management server, user with write access to only one node can potentially acquire information about nodes to which he normally does not have access. For example, without additional security checks user with write access to node `A` and no access to node `B` can create transformation script for DCI on node `A` and use FindNodeObject function to access node `B` and get information about it, thus breaking security settings.

To prevent such scenario, all NXSL functions capable of accessing NetXMS objects requires "current node" object to be provided. Reference to object being searched will only be returned if node object supplied as "current node" is in trusted nodes list of target object. For example, if variable `$node` in script refers to `NODE1`, and `FindNodeObject($node, "NODE2")` called, `NODE1` must be added to list of trusted nodes for `NODE2`. In most places (transformation script, event processing policy, etc.) predefined variable `$node` exists, which refers to node object on behalf of which script is being executed. It will be event source for event processing policy script, DCI owner for transformation script, and so on.

For environments where such strict security checks are not required (for example, all users have read access to all nodes), they can be disabled to simplify configuration. Enforcement of trusted nodes checking controlled by server's configuration variable `CheckTrustedNodes`. By default it is set to `1` and check of trusted nodes is enforced. To disable it, server's configuration variable `CheckTrustedNodes` must be set to `0`. The server restart is required to make this change effective.

Language syntax

Built-in Types

The following sections describe the standard types that are built into the interpreter.

NXSL is loose typed programming language. The system will automatically determine each variable type, assign a certain type to a variable and convert a variable type from one to another, if necessary. For example, a result for `3 + "4"` will be `7`, because the system will automatically convert `"4"` string into an integer. In case if the system is not able to automatically convert a line into an appropriate integer, the operation will result in a runtime error.

NXSL supports the following variable types:

- integer (32 bit),
- unsigned integer (32 bit),
- integer (64 bit), unsigned integer (64 bit),
- floating-point number,
- string,
- boolean,
- array,
- hash map,
- object.

In addition to that, NXSL also supports a special variable type – `NULL`. This value represents a variable with no value. `NULL` is the only possible value of type `NULL`. An attempt to perform any type of arithmetical or string operations with `NULL` variable will result in system runtime error.

It is possible to manually convert variable to a certain type, using a special function, named depending on the variable type. For example, `string(4)`. That way it is also possible to convert `NULL` type variables. Therefore, to avoid runtime errors while processing `NULL` type variables, it is advised to use manual conversion.

NXSL does not require setting variable type beforehand. The only exception to this is arrays. In case if an array is required, operator `array` defines its subsequent variables as arrays. Accessing variable which was not previously assigned will return `NULL` value.

Although NXSL has object type variables, it is not an object-oriented language. It is not possible to define classes or create objects at script level – only in extensions written in C++. Object type variables are used to return information about complex NetXMS objects, like nodes or events, in a convenient way. Please note that assigning object type variables actually holds reference to an object, so assigning object value to another variable does not duplicate actual object, but just copy reference to it.

To get a human-readable representation of a variable or expression type for debugging, use the

`typeof()` function, and to get a class name for object type variables, use `classof()` function.

String literals

A string literal is where you specify the contents of a string in a program. There are few different string literal types:

- "string" - string literal where some characters preceded by a backslash have special meaning. If you need to insert double quote character you should prepend it with backslash.
- 'string' - string literal with backslash being an ordinary character without special meaning. You cannot insert single quote character into such string literal.
- """ Multi

  line

  string""" - string literal that can span multiple lines with backslash being an ordinary character.

Truth Value Testing

Any variable can be tested for truth value, for use in an `if` or `while` condition or as operand of the Boolean operations below. The following values are considered false:

- `false`
- `NULL`
- zero of any numeric type, for example, `0`, `0.0`, `0j`.

All other values are considered `true` — so objects of various types and arrays are always `true`.

Operations and built-in functions that have a Boolean result return boolean type. However, there's an exception to this due to optimizations related to short-circuit evaluation of `or` and `and` operators. For `or`, if first operand not boolean type, but it's value is considered `true`, that operand will be returned as result. For `and`, if first operand is considered `false`, that operand will be returned as result. This will not cause issues in subsequent logical operations, as returned value will be correctly considered as `true` or `false`; But when printing the result, it will not be converted to Boolean, so conversion using boolean() function might be needed.

For example:

```
a = %(1,2,3);
b = true;
c = a or b;
println(c);  // will print "[1, 2, 3]" because first operand was returned by or
operation
if (c) println("TRUE");  // will print "TRUE" as the array contained in c is
considered as true
```

```
a = 0;
```

```
b = true;
c = a and b;
println(c); // will print "0" because first operand was returned by and operation
println(boolean(c));  // will print "false"
```

Variables

Variables in NXSL behave the same way as variables in most popular programming languages (C, C++, etc.) do, but in NXSL you don't have to declare variables before you use them.

Scope of a variable can be either global (visible in any function in the script) or local (visible only in the function within which it was defined). Any variable is by default limited to the local function scope. Variable can be declared global using `global` operator.

For example:

```
x = 1;
myFunction();

function myFunction()
{
    println("x=" . x);
}
```

This script will cause run time error `Error 5 in line 6: Invalid operation with NULL value`, because variable `x` is local (in implicit main function) and is not visible in function `myFunction`. The following script will produce expected result (prints `x=1`):

```
global x = 1;
myFunction();

function myFunction()
{
    println("x=" . x);
}
```

Function Declaration

A function is a named code block that is generally intended to process specified input values into an output value, although this is not always the case. For example, the `trace()` function takes variables and static text and prints the values into server log. Like many languages, NXSL provides for user-defined functions. These may be located anywhere in the main program or loaded in from other scripts via the use keywords.

To define a function, you can use the following form:

function *NAME* **(** *ARGUMENTS* **) BLOCK**

where `NAME` is any valid identifier, `ARGUMENTS` is optional list of argument names, and `BLOCK` is code block.

To call a function you would use the following form:

NAME **(** *LIST* **)**

where `NAME` is identifier used in function definition, and `LIST` is an optional list of expressions passed as function arguments.

To give a quick example of a simple subroutine:

```
function message()
{
    println("Hello!");
}
```

Function Arguments

The first argument you pass to the function is available within the function as `$1`, the second argument is `$2`, and so on. For example, this simple function adds two numbers and prints the result:

```
function add()
{
    result = $1 + $2;
    println("The result was: " . result);
}
```

To call the subroutine and get a result:

```
add(1, 2);
```

If you want named arguments, list of aliases for `$1`, `$2`, etc. can be provided in function declaration inside the brackets:

```
function add(numberA, numberB)
{
    result = numberA + numberB;
    println("The result was: " . result);
}
```

If parameter was not provided at function call, value of appropriate variable will be `NULL`.

Another option to name parameters is to provide named parameters inside of function. In this case leave braces of function empty and add names to function call. In this case parameters will be available in function as $parameterName.

Example:

```
func(param2: "text2", param1: "text1");
return 0;

function func()
{
    println($param1); //Will print "text1"
    println($param2); //Will print "text2"
}
```

The arguments received by script when it was launched are available in a global variable $ARGS, which is an array that contains all arguments. First argument available as $ARGS[1];

Return Values from a Function

You can return a value from a function using the `return` keyword:

```
function pct(value, total)
{
    return value / total * 100.0;
}
```

When called, return immediately terminates the current function and returns the value to the caller. If you don't specify a value in `return` statement or function ends implicitly by reaching end of function's block, then the return value is `NULL`.

Script entry point

NXSL handles script entry in 2 ways:

- Explicit main() function
- Implicit $main() function

If an explicitly defined main() exists, it will be called.

If an explicit main() doesn't exist, an implicit $main() function will be created by the script interpreter and the script will enter at the $main() function.

The $main() function is constructed from code that is not a part of any other functions.

Calling library script functions

You can call functions from scripts that are stored in Script Library. One way is to use the `use` keyword accompanied by the name of the script:

```
use my_math_library;
println( add(1, 2) );
```

The other was is shown in this example:

```
println( my_math_library::add(1, 2) );
```

Strings

Strings are not objects, it's a separate variable type. However, strings have methods and attributes described below

String attributes

`isEmpty ⇒ Boolean`

Returns "true" if string is empty or "false" otherwise.

Example

```
s = "";
println(s->isEmpty); // prints "true"
```

`length ⇒ Integer`

Returns number of characters in the string.

Example

```
s = "1234567890";
println(s->length); // prints '10'
```

String methods

`compareTo(string) ⇒ Integer`

Compares two strings lexicographically (alphabetical order). Returns `-1` if the argument is a string lexicographically greater than this string, `1` if the argument is a string lexicographically less than this string or `0` if the argument is a string lexicographically equal to this string.

The is a difference to `==` comparison operator if strings contain numbers because values are converted to numberic type prior to comparison.

Example

```
println("a"->compareTo("c")); // prints "-1"
println("c"->compareTo("a")); // prints "1"
println("a"->compareTo("aaa")); // prints "-1"
println("aaa"->compareTo("a")); // prints "1"

println("100"->compareTo("100.0")); // prints "-1"
println("100" == "100.0"); // prints true
```

`compareToIgnoreCase(string) ⇒ Integer`

Same as `compareTo`, but ignoring the case.

Example

```
println("aaa"->compareToIgnoreCase("AAA")); // prints "0"
```

`contains(string) ⇒ Boolean`

Returns `true`` if this string contains the argument string or `false` otherwise.

Example

```
println("aaa"->contains("a")); // prints "true"
```

`endsWith(string) ⇒ Boolean`

Returns `true`` if this string ends with the argument string or `false` otherwise.

Example

```
println("abc"->endsWith("d")); // prints "false"
println("Just a sentence"->endsWith("a sentence")); // prints "true"
```

`equalsIgnoreCase(string) ⇒ Boolean`

Returns `true` if argument string is equal to this string ignoring the case or `false` otherwise.

Example

```
println("abc"->equalsIgnoreCase("ABC")); // prints "true"
```

`indexOf(string) ⇒ Integer`

Returns index of first occurence of argument string in this string or `-1` if the argument is not a substring of the string.

Example

```
println("ABC-DEF-GHI"->indexOf("-")); // prints "3"
println("ABC-DEF-GHI"->indexOf("ABC")); // prints "0"
```

```
println("ABC-DEF-GHI"->indexOf("JKL")); // prints "-1"
```

`lastIndexOf(string) ⇒ Integer`

Returns index of last occurence of argument string in this string or `-1` if the argument is not a substring of the string.

Example

```
println("ABC-DEF-GHI"->lastIndexOf("-")); // prints "7"
println("ABC-DEF-GHI"->lastIndexOf("ABC")); // prints "0"
println("ABC-DEF-GHI"->lastIndexOf("JKL")); // prints "-1"
```

`left(numberOfCharacters, paddingCharacter) ⇒ String`

Returns left `numberOfCharacters` of this string. If string length is less then `numberOfCharacters`, result will be padded on the right. `paddingCharacter` is optional, if not specified, space character will be used for padding.

Example

```
println("ABCDEFGHI"->left(2)); // prints "AB"
println("123"->left(5)); // prints "123  "
println("123"->left(5, "_")); // prints "123__"
```

`replace(whatToReplace, replaceBy) ⇒ String`

Returns string where all occurencies of `whatToReplace` are replaced with `replaceBy`.

Example

```
println("A B C A D K L"->replace("A", "<A>")); // prints "<A> B C <A> D K L"
```

`right(numberOfCharacters, paddingCharacter) ⇒ String`

Returns right `numberOfCharacters` of this string. If string length is less then `numberOfCharacters`, result will be padded on the left. `paddingCharacter` is optional, if not specified, space character will be used for padding.

Example

```
println("ABCDEFGHI"->right(2)); // prints "HI"
println("123"->right(5)); // prints "  123"
println("123"->right(5, "_")); // prints "__123"
```

`split(separator) ⇒ Array`

Split string into array of strings at given separator.

Example

```
println("ABC--DEF--GHI"->split("--")); // prints "[ABC, DEF, GHI]"
```

`startsWith(string) ⇒ Boolean`

Returns `true` if this string starts with the argument string or `false` otherwise.

Example

```
println("abc"->startsWith("d")); // prints "false"
println("Just a sentence"->startsWith("Just a")); // prints "true"
```

`substring(position, numberOfCharacters) ⇒ String`

Returns substring of this string starting from `position` and containing `numberOfCharacters`.

Example

```
println("ABCDEFGHIJK"->substring(0,3)); // prints "ABC"
println("ABCDEFGHIJK"->substring(6,3)); // prints "GHI"
println("ABCDEFGHIJK"->substring(6,10)); // prints "GHIJK"
```

`toLowerCase() ⇒ String`

Converts this string to lowercase.

Example

```
println("ABC def"->toLowerCase()); // prints "abc def"
```

`toUpperCase() ⇒ String`

Converts this string to uppercase.

Example

```
println("ABC def"->toUpperCase()); // prints "ABC DEF"
```

`trim() ⇒ String`

Returns this string with whitespace from both sides removed.

Example

```
println("|" . "  ABC  "->trim() . "|") // prints "|ABC|"
```

`trimLeft() ⇒ String`

Returns this string with whitespace from left side removed.

Example

```
println("|" . "  ABC  "->trimLeft() . "|") // prints "|ABC   |"
```

`trimRight() ⇒ String`

Returns this string with whitespace from right side removed.

Example

```
println("|" . "  ABC  "->trimRight() . "|") // prints "|   ABC|"
```

Arrays

An array in NXSL is actually an ordered map. A map is a type that associates `values` to `keys`. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, and probably more. Nested arrays are supported, so elements of an array can be themselves arrays.

A `key` is 32-bit signed integer. When an array is created, its size is not specified and its map can have empty spots in it. For example, an array can have a element with a `0` key and an element with `4` key and no keys in-between. Attempting to access an array key which has not been defined is the same as accessing any other undefined variable: the result will be `NULL`.

Arrays are not objects, it's a separate variable type. However, arrays have methods and attributes described below.

Array elements can be accessed using [`index`] operator. For example, to access element with index `3` of array `a` you should use

```
a[3];
```

To get subarray from the array use `[a:b]` operator. This operator returns subarray of an array from the element with index `a` inclusive till the element with index `b` exclusive. If `a` is omitted then subarray will be taken from the start of the array and if `b` is omitted then subarray will be taken till the end of the array.

Example:

```
a = %(1, 2, 3, 4);
a2 = a[1:3]; // a2 will be %(2, 3)
a3 = a[1:]; // a3 will be %(2, 3, 4)
```

Array Initialization

New array can be created in two ways. First is to use `array` operator. This statement will create empty array and assign reference to it to variable `a`.

```
array a;
```

You can then assign values to the array. Please note arrays in NXSL are sparse, so indices can contain gaps.

```
array a;
a[1] = 1;
a[2] = 2;
a[260] = 260;
println(a[1]); // will print 1
println(a); // will print "[1, 2, 260]"
```

Second way is to use %( ) construct to create array already populated with values.

This statement will create array with four elements at positions 0, 1, 2, and 3, and assign reference to this array to variable a.

```
// no need to use "array a;" here, since we are creating it directly
a = %(1, 2, 3, 4);

println(a[0]); // will actually print 1, since 1 is the 0th member
println(a); // will print "[1, 2, 3, 4]"
```

Array initialization can also be used directly in expressions, like this:

```
function f()
{
    return %(2, "text", %(1, 2, 3));
}
```

In this example function `f` returns array of 3 elements - number, text, and another array of 3 numeric elements.

Array attributes

`maxIndex ⇒ Integer`

Returns highest index in the array.

Example

```
a = %(1, 2, 3);
println(a->maxIndex); // prints '2'
println(a[a->maxIndex]); // prints '3'
```

`minIndex ⇒ Integer`

Returns lowest index in the array.

Example

```
a = %(1, 2, 3);
println(a->minIndex); // prints '0'
```

`size ⇒ Integer`

Returns number of elements in the array.

Example

```
a = %(1, 2, 3);
println(a->size); // prints '3'
```

Array methods

`append(newElement) ⇒ Integer`

Appends new element to the array. Returns highest index in the array - that's index of the appended element.

Example

```
a = %("a","b","c");
a->append("d");
println(a); // prints '[a, b, c, d]'
```

`appendAll(anotherArray) ⇒ Integer`

Appends elements of `anotherArray` to the array. Returns highest index in the array.

Example

```
a = %(1,2);
b = %(3,4);
a->appendAll(b);
println(a); // prints '[1, 2, 3, 4]'
```

`insert(index, newElement) ⇒ void`

Inserts new element to the array at `index`. Indexes of existing elements that had index greater or equal to `index` are incremented.

Example

```
array a;
a[0] = "aaa";
a[10] = "ccc";
```

```
a->insert(5, "bbb");
println(a[0]);  // prints "aaa"
println(a[5]);  // prints "bbb"
println(a[11]);  // prints "ccc" - because of the insert operation this element's
index is now 11.
```

`insertAll(index, anotherArray) ⇒ void`

Inserts elements of `anotherArray` to the array at `index`. Indexes of existing elements that have index greater or equal to `index` are incremented.

Example

```
a = %(1,2);
b = %(3,4);
a->insertAll(1,b);
println(a); // prints '[1, 3, 4, 2]'
```

`pop() ⇒ Element with highest index`

Removes element with highest index from the array. Using `push(value)` and `pop()` methods it's possible to use array as a stack. Or, using `insert(0,value)` and `pop()`, array will work as FIFO queue.

Example

```
a = %();
a->push("one");
a->push("two");
println(a->pop());
println(a->pop());
```

`push(newElement) ⇒ Integer`

Same as `append()`.

`remove(index) ⇒ void`

Removes element at specified `index`. Indexes of elements that have index greater or equal to `index` are decremented.

Example

```
a = %(1,2,3);
a->remove(0);
println(a);
```

Array conversion

Array can be converted to string. `string(array)` function is used to get string representation of array. The string representation consists of a list of the array's elements, enclosed in square

brackets (“[]”). Adjacent elements are separated by the characters “, ” (a comma followed by a space).

Printed array is automatically converted to string.

```
a = %(1, 2, 3, 4, 5, 6, 7);
println(a); // will print "[1, 2, 3, 4, 5, 6, 7]"
println(a . " is an array"); // will print "[1, 2, 3, 4, 5, 6, 7] is an array"
println(%("one", "two")); // will print "[one, two]"
println(%(2, "text", %(1, 2, 3))); // will print "[2, text, [1, 2, 3]]"
```

Hash maps

Hash map allows to store data values in key:value pairs. A key is string. Numeric type can also be supplied as key, but it will be internally converted to string. Hash map cannot have two items with the same key. The values can be of any data type, including null, objects, arrays or hash maps.

Hash maps are not objects, it’s a separate variable type. However, hash maps have methods and attributes described below.

Array elements can be accessed using [key] operator. For example, to access element with key key of hash map h you should use

```
h["key"];
```

Hash Map Initialization

This statement will create an empty hash map and assign reference to in to variable h:

```
h = %{};
```

It’s also possible to create hash map already populated with values, e.g.:

```
h = %{"key":123, "another_key":456};
```

Hash Map Attributes

`keys ⇒ Array`

Returns array with keys of items in the hash map.

Example

```
h = %{100:"value1", 101:"value2"};
println(h->keys); // prints '[100,101]'
```

`size ⇒ Integer`

Returns number of items in the hash map.

Example

```
h = %{"a":null, "b":null, "c":null};
println(h->size); // prints '3'
```

`values ⇒ Array`

Returns array with values of items in the hash map.

Example

```
h = %{"key1":123, "key2":456};
println(h->keys); // prints '[123,456]'
```

Hash Map Methods

`contains(key) ⇒ Boolean`

Returns `true`, if hash map contains specified key or `false` otherwise.

Example

```
h = %{"key1":123, "key2":456};
println(h->contains("key2")); // prints 'true'
```

`remove(key) ⇒ void`

Removes item with specified `key`.

Example

```
h = %{"key1":123, "key2":456};
h->remove("key1");
println(h); // prints '{key2=456}'
```

Hash Map Conversion

Hash Map can be converted to string. `string(hash-map)` function is used to get string representation of hash map. The string representation lists all key-value pairs enclosed in curly brackets (“{}”). Value is separate from the key with equals sigh ("="). Items are separated by the characters “, ” (a comma followed by a space).

Printed array is automatically converted to string.

```
h = %{"key1":123, "key2":456};
println("This is a hash map: " . string(h));
```

```
println("Or we can just print it this way: " . h);
```

Operators

An operator is something that you feed with one or more values, which yields another value.

Arithmetic Operators

Example	Name	Result
`-a`	Negation	Opposite of `a`
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference between `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	Division	Quotient of `a` and `b`
`a % b`	Modulus	Remainder of `a` divided by `b`

The division operator (`/`) returns a float value unless the two operands are integers (or strings that get converted to integers) and the numbers are evenly divisible, in which case an integer value will be returned.

Calling modulus on float operands will yield runtime error.

Assignment Operator

The assignment operator is `=`, which means that the left operand gets set to the value of the expression on the rights (that is, "gets set to").

Bitwise Operators

Example	Name	Result
`~ a`	Not	Bits that are set in `a` are unset, and vice versa.
`a & b`	And	Bits that are set in both operand are set.
`a | b`	Or	Bits that are set in either operand are set.
`a ^ b`	Xor	Bits that are set in only one operand are set.
`a << b`	Shift left	Shift the bits of `a` for `b` steps to the left (each step equals "multiply by two").

Example	Name	Result
`a >> b`	Shift right	Shift the bits of `a` for `b` steps to the right (each step equals "divide by two").

Comparison Operators

Comparison operators allow you to compare two values.

Example	Name	Result
`a == b`	Equal	`TRUE` if `a` is equal to `b`.
`a != b`	Not equal	`TRUE` if `a` is not equal to `b`.
`a < b`	Less than	`TRUE` if `a` is strictly less than `b`.
`a > b`	Greater than	`TRUE` if `a` is strictly greater than `b`.
`a <= b`	Less than or equal to	`TRUE` if `a` is less than or equal to `b`.
`a >= b`	Greater than or equal to	`TRUE` if `a` is greater than or equal to `b`.
`a ~= b`	Match	Array containing full match of `b` and capture groups, if any. If nothing was matched, false (boolean) is returned. Capture groups are also assigned to special variables $1, $2, $3, etc. See see Regular expressions for additional information.
`a match b`	Match	Same as `a ~= b`
`a imatch b`	Match (case insensitive)	Same as `a ~= b` or `a match b`, but matching is case insensitive.
`a like b`	Like	Compare string value to a pattern using wildcard characters. Two wildcard characters are supported: `*` - represents zero, one or multiple characters. `?` - represents any single character. Wildcard characters can be used in combinations.
`a ilike b`	Like (case insensitive)	Same as `a like b`, but comparison is case insensitive.

Example:

```
println("aaa bbb ccc" ~= "b+") // prints "[bbb]"
println("Username: John" ~= "Username: (\w+)"); // prints "[Username: John, John]"

println("abc" like "?bc*"); // prints "true"
```

Note that strings which actually contain number are converted to numeric type prior to comparison. So, for example:

```
s1 = "1";
s2 = "1.0";
i = 1;
println(s1 == s2); // prints "true"
println(s1 == i); // prints "true"
```

Incrementing/Decrementing Operators

NXSL supports C-style pre- and post-increment and decrement operators.

Example	Name	Result
`++a`	Pre-increment	Increments `a` by one, then returns `a`.
`a++`	Post-increment	Returns `a`, then increments `a` by one.
`--a`	Pre-decrement	Decrements `a` by one, then returns `a`.
`a--`	Post-decrement	Returns `a`, then decrements `a` by one.

Logical Operators

Example	Name	Result
`! a`	Not	`TRUE` if `a` is not `TRUE`.
`not a`	Not	Same as above. `TRUE` if `a` is not `TRUE`.
`a && b`	And	`TRUE` if both `a` and `b` is `TRUE`.
`a and b`	And	Same as above. `TRUE` if both `a` and `b` is `TRUE`.
`a || b`	Or	`TRUE` if either `a` or `b` is `TRUE`.
`a or b`	Or	Same as above. `TRUE` if either `a` or `b` is `TRUE`.

String Operators

Example	Name	Result
`.`	Concatenation operator	Returns the concatenation of its right and left arguments.
`.=`	Concatenating assignment operator	Appends the argument on the right side to the argument on the left side.
`[a:b]`	Substring operator	Returns substring of a string from the character with index `a` inclusive till the character with index `b` exclusive. Example: "1234"[1:3] will be "23". If `a` is omitted then substring will be taken from the start of the string and if `b` is omitted then substring will be taken till the end of the string.

Control structures

Any NXSL script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing (an empty statement). Statements usually end with a semicolon. In addition, statements can be grouped into a statement-group by encapsulating a group of statements with curly braces. A statement-group is a statement by itself as well. The various statement types are supported:

- if
- else
- while
- do-while
- for
- break
- continue
- switch
- with
- return
- exit

if

The `if` construct is one of the most important features of many languages. It allows for conditional execution of code fragments. NXSL features an `if` structure that is similar to that of C:

```
if (expr)
    statement
```

else

Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is what `else` is for. `else` extends an `if` statement to execute a statement in case the expression in the `if` statement evaluates to `FALSE`. The `else` statement is only executed if the `if` expression evaluated to `FALSE`.

while

`while` loops are the simplest type of loop in NXSL. They behave just like their C counterparts. The basic form of a `while` statement is:

```
while (expr)
    statement
```

The meaning of a `while` statement is simple. It tells NXSL to execute the nested statement(s) repeatedly, as long as the `while` expression evaluates to `TRUE`. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration.

do-while

`do-while` loops are very similar to `while` loops, except the truth expression is checked at the end of each iteration instead of in the beginning. The main difference from regular `while` loops is that the first iteration of a `do-while` loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it may not necessarily run with a regular `while` loop (the truth expression is checked at the beginning of each iteration, if it evaluates to `FALSE` right from the beginning, the loop execution would end immediately).

for

`for` loops are the most complex loops in NXSL. They behave in two different ways: like their C counterparts or in Java way. The syntax of a `for` loop is:

```
for (expr1; expr2; expr3)
    statement

for (varName : array)
    statement
```

The first expression (`expr1`) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, `expr2` is evaluated. If it evaluates to `TRUE`, the loop continues and the nested statement(s) are executed. If it evaluates to `FALSE`, the execution of the loop ends.

At the end of each iteration, `expr3` is evaluated (executed).

In the second example for cycle will call `statement` for each element in array. Element will be available as `varName`.

break

`break` ends execution of the current `for`, `while`, `do-while` or `switch` structure.

continue

`continue` is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.

switch

The `switch` statement is similar to a series of `if` statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the `switch` statement is for.

Example:

```
switch (input)
{
  case "1":
    trace(0,"Input is 1");
    break;
  case "2":
    trace(0,"Input is 2");
    break;
  default:
    trace(0, "Input is unknown");
}
```

The `switch` statement also allows to check ranges:

```
switch (input)
{
  case 1:
    trace(0,"Input is 1");
    break;
  case 2:
    trace(0,"Input is 2");
    break;
  case 3...7:
```

```
    trace(0,"Input is from 3 till 7");
    break;
  default:
    trace(0, "Input is unknown");
}
```

with

With statement is made to make the code cleaner and much more readable and to expose variable section to global scope for "Object query" Dashboard element. This statement consists of 2 parts: variable declaration (optional) and expression.

Structure:

```
with
  var = {code},
  ...
  var = {code}
expression
```

Example for "Object query" Dashboard element. This example will filter out only nodes that are unreachable and will create 2 variables as data providers for columns: time node is down since and oldest alarm time.

```
with
  _down = { return SecondsToUptime(time() - downSince); },
 _oldestAlarm = {
  oldestAlarmTime = 99999999999;
  for (a : $node->alarms) {
   oldestAlarmTime = min(oldestAlarmTime, a->creationTime);
  }
  return strftime("%Y-%m-%d %H:%M", oldestAlarmTime);
 }
type == NODE and state & NodeState::Unreachable
//In Object query object attributes are available just using name.
//Like state ($node->state in other scripts)
```

return

If called from within a function, the `return` statement immediately ends execution of the current function, and returns its argument as the value of the function call. Calling `return` from `main()` function (either explicitly or implicitly defined) is equivalent of calling `exit`.

exit

The `exit` statement immediately ends execution of the entire script, and returns its argument as script execution result.

Expressions

The simplest yet most accurate way to define an expression is "anything that has a value".

The most basic forms of expressions are constants and variables. When you type `a = 5`, you're assigning `5` into `a`. `5`, obviously, has the value 5, or in other words `5` is an expression with the value of 5 (in this case, `5` is an integer constant).

Slightly more complex examples for expressions are functions. Functions are expressions with the value of their return value.

NXSL supports the following value types: integer values, floating point values (float), string values and arrays. Each of these value types can be assigned into variables or returned from functions.

Another good example of expression orientation is pre- and post-increment and decrement. You be familiar with the notation of `variable+` and `variable--`. These are increment and decrement operators. In NXSL, like in C, there are two types of increment - pre-increment and post-increment. Both pre-increment and post-increment essentially increment the variable, and the effect on the variable is identical. The difference is with the value of the increment expression. Pre-increment, which is written `+variable`, evaluates to the incremented value. Post-increment, which is written `variable++` evaluates to the original value of variable, before it was incremented.

A very common type of expressions are comparison expressions. These expressions evaluate to either `FALSE` or `TRUE`. NXSL supports `>` (bigger than), `>=` (bigger than or equal to), `=` (equal), `!=` (not equal), `<` (less than) and `<=` (less than or equal to). These expressions are most commonly used inside conditional execution, such as `if` statements.

The last example of expressions is combined operator-assignment expressions. You already know that if you want to increment `a` by 1, you can simply write `a+` or `+a`. But what if you want to add more than one to it, for instance 3? In NXSL, adding 3 to the current value of `a` can be written `a += 3`. This means exactly "take the value of `a`, add 3 to it, and assign it back into `a` ". In addition to being shorter and clearer, this also results in faster execution. The value of `a += 3`, like the value of a regular assignment, is the assigned value. Notice that it is NOT 3, but the combined value of `a` plus 3 (this is the value that's assigned into `a`). Any two-place operator can be used in this operator-assignment mode.

Short-circuit evaluation

Short-circuit evaluation denotes the semantics of some Boolean operators in which the second argument is only executed or evaluated if the first argument does not suffice to determine the value of the expression: when the first argument of the AND function evaluates to false, the overall value must be false; and when the first argument of the OR function evaluates to true, the overall value must be true. NXSL uses short-circuit evaluation for `&&` and `||` boolean operators. This feature permits two useful programming constructs. Firstly, if the first sub-expression checks whether an expensive computation is needed and the check evaluates to false, one can eliminate expensive computation in the second argument. Secondly, it permits a construct where the first expression guarantees a condition without which the second expression may cause a run-time error. Both are illustrated in the following example:

```
if ((x != null) && ((trim(x) == "abc") || (long_running_test(x)))
   do_something();
```

Without short-circuit evaluation, `trim(x)` would cause run-time error if `x` is `NULL`. Also, long running function will only be called if condition (`trim(x) == "abc"`) will be false.

Regular expressions

Since version 3.0, regular expression engine is changed to PCRE (Perl compatible). Syntax can be checked with `pcregrep`, perl itself or on regex101.com (select PCRE flavour).

Comments

Tutorial

Syntactically, NXSL looks similar to Perl or C. Here's simple NXSL program:

```
/* sample program */
function main()
{
    println("Hello!");
    return 0;
}
```

This program will print word `Hello` on screen.

Also, keep in mind that you are free to choose your own formatting style. E.g. the above could have been written as:

```
/* sample program */ function main(){println("Hello!");return 0;}
```

Now we'll analyze this program:

```
/* sample program */
```

Everything inside `/* */` is considered a comment and will be ignored by interpreter. You can enclose comments, like below:

```
/* comment /* another comment */ still comment */
```

You can also use single line comments:

```
x = 1; // everything between two slashes and end of line is a comment
```

Now onto next line:

```
function main()
{
}
```

This is a function definition. A function is a part of a program that can be called by other parts of the program. A function definition always has the following form:

```
function name(parameters)
{
```

```
    // the function code goes here
}
```

The function can return a value to the caller and accept zero or more parameters.

The function name follows the rules for all names (formally: identifiers): it must consist entirely of letters (uppercase and lowercase are different!), digits, underscores (_) and dollar signs ($), but may not begin with a digit. Please note that most special identifiers starts with dollar sign ($), so it is recommended not to start your identifiers with it.

First line in function code looks like

```
println("Hello!");
```

In this line, `println` is an embedded operator which prints given string to standard output with carriage return, and `"Hello!"` is a string we want to print. Please note semicolon at the end of line – it's a separator between operators. Each operator should end with semicolon.

The next, and final, line of our small program is:

```
return 0;
```

`return` is another built-in operator which exits the function and sets it's return value.

Function Reference

Strings

ArrayToString()

`ArrayToString(array,separator) ⇒ String`

Convert array to string

Parameters

array	Array	Array with elements to concatenate
separator	String	Separator between array elements

Return

Concatenated string

Example

```
a = %(1, 2, 3, 4);
b = ArrayToString(a, ";");
println(b); // will print "1;2;3;4"
```

chr()

```
chr(code) => void
```

Return a character from it's UNICODE code.

Parameters

code	Integer	A character's UNICODE code.

Return

A string consisting of single character.

Example

```
chr(50)  //Will return "P"
```

d2x()

```
d2x(number, padding=0) => String
```

Convert decimal `devValue` to hex string with optional left-padding with zeroes.

Parameters

number	Input value.
padding	Optional argument specifying target string length.

Return

Hex string.

Example

```
>>> d2x(1234)
4D2
>>> d2x(1234, 8)
000004D2
```

format()

```
format(number, width, precision) => String
```

Formats a numeric value.

Parameters

number	Number	The numeric value to format.
width	Number	Minimum number of characters. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values — depending on whether the width is negative (for left alignment) or positive (for right alignment) — until the minimum width is reached. The width specification never causes a value to be truncated.
precision	Number	The number of decimal places. Floating point value will be rounded accordingly.

Return

Formatted numeric value.

Example

```
format(3.7, 7, 2)       //  "   3.70"
format(3.7, -7, 2)      //  "3.70   "
format(5.7278, 1, 2)    //  "5.73"
format(5.7278, 1, 0)    //  "6"
```

inList()

```
inList(string, separator, token) => Boolean
```

Split input `string` by `separator` into elements and compare each element with `token`.

Parameters

string	Input string.
separator	Elements separator.
token	Pattern to compare with.

Return

`True` if token is found in input string.

Example

```
>>> inList("1,2,3", ",", "1")
true
>>> inList("ab|cd|ef", "|", "test")
false
```

ord()

```
ord(character) => Integer
```

Convert a character into it's ASCII/Unicode value.

Only processes one character.

Parameters

character	String	Character

Return

An ASCII/Unicode value

Example

```
println(ord("a")); //Will pritn 97
println(ord("abc")); //Will pritn 97
```

x2d()

```
x2d(hexValue) => Integer
```

Convert hexadecimal string to decimal value.

Parameters

hexValue	Input value.

Return

Converted value.

Example

```
>>> x2d("4D2")
1234
```

Math

abs()

`abs(number) ⇒ Number`

Returns the absolute value of the number.

Parameters

number	Number	Input value.

Return

Absolute value of the input.

Example

```
>>> abs(12.3)
12.3
>>> abs(-0.307)
0.307
```

acos()

`acos(number) ⇒ Number`

Calculates arc cosine x

Parameters

number	Number	Real number x, with −1 ≤ x ≤ 1

Return

The angle in radians whose cosine is x

Example

```
acos(-1) //Result: 3.141593
acos(0.5) //Result: 1.047198
```

asin()

`asin(number) ⇒ Number`

Calculates arc sine x

Parameters

number	Number	Real number x, with −1 ≤ x ≤ 1

Return

The angle in radians whose sine is x

Example

```
asin(1) //Result: 1.570796
asin(0.5) //Result: 0.523599
```

atan()

```
atan(number) => Number
```

Calculates arc tangent x

Parameters

number	Number	Real number x, with −1 ≤ x ≤ 1

Return

The angle in radians whose arc tangent is x

Example

```
atan(1) //Result: 0.785398
atan(0.5) //Result: 0.463648
```

atan2()

```
atan2(number1, number2) => Number
```

Calculates 2-argument arc tangent

Parameters

number1	Number	Real number x, with −1 ≤ x ≤ 1
number2	Number	Real number x, with −1 ≤ x ≤ 1

Return

The angle in radians

Example

```
atan2(1, 0.5) //Result: 1.107149
atan2(0.5, 1) //Result: 0.463648
```

ceil()

```
ceil(input) => Integer
```

Round up value.

Parameters

input	Input value.

Return

Value round up to nearest integer.

Example

```
>>> ceil(2.3)
3.0
>>> ceil(3.8)
4.0
>>> ceil(-2.3)
-2.0
>>> ceil(-3.8)
-3.0
```

cos()

```
cos(x) => Number
```

Calculates cosine from given angle in radians.

Parameters

x	Number	Angle in rad

Return

Result of cosine for this angle

Example

```
print(cos(0.5)); //will print 0.877583
```

cosh()

```
cosh(x) => Number
```

Calculates hyperbolic cosine x

Parameters

x	Number	Angle in rad

Return

Result of hyperbolic cosine for this angle

Example

```
print(cosh(0.5)); //will print 0.877583
```

exp()

```
exp(input) => Float
```

Computes `e**x`, the base-e exponential.

Parameters

input	Input number.

Return

Result of the exponential function

Example

```
>>> exp(2)
7.3890561
```

floor()

```
floor(input) => Integer
```

Round down value.

Parameters

input	Input value.

Return

Value round down to nearest integer.

Example

```
>>> floor(2.3)
2
>>> floor(3.8)
3
>>> floor(-2.3)
-3
>>> floor(-3.8)
-4
```

log()

```
log(x) => Number
```

Calculates natural logarithm

Parameters

x	Number	Number to calculate natural logarithm of

Return

Natural logarithm of x.

Example

```
ptintln log(2); // Will print "0.693147"
```

log10()

```
log10(x) => Number
```

Calculates logarithm of given value to base 10.

Parameters

x	Number	Number to calculate logarithm of

Return

Logarithm of x to base 10.

Example

```
pritnl log10(2); // Will print "0.301030"
```

max()

```
max(number1 ,number2], ...) => void
```

Returns maximal value from a list of values.

Parameters

numbers	Integer	Numbers separated by comma

Return

Maximal value of numbers.

Example

```
max(2, 3, 4, 8); //Will print "8"
```

min()

```
min(number1 ,number2, ...) => Number
```

Returns minimal value from a list of values.

Parameters

numbers	Number	Coma separated numbers

Return

Minimal value of numbers.

Example

```
println(min(2, 3, 4, 8)); // Will print "2"
```

pow()

```
pow(x, y) => Number
```

Calculates x raised to the power of y.

Parameters

x	Number	Initial value.
y	Number	Power.

Return

x raised to the power of y.

Example

```
println(pow(2, 3)); //Will print "8"
```

round()

```
round(x, precision) => Number
```

Round floating point value to the nearest integral value or floating point value with given precision.

Parameters

x	Number	Floating point value.
precision	Integer	Optional number of decimal places to be left. If omitted or set to 0, x will be rounded to integral value.

Return

The integral value that is closest to x if precision is omitted or set to 0, or floating point value rounded to have given number of decimal places.

Example

```
println(round(2.3)); // Will print "2"
println(round(3.8)); // Will print "4"
println(round(-2.3)); // Will print "-2"
println(round(-3.8)); // Will print "-4"
println(round(2.378, 2)); // Will print "2.38"
println(round(2.378, 1)); // Will print "2.4"
```

sin()

```
sin(x) => Number
```

Calculates sine from given angle in radians.

Parameters

x	Number	Angle in radian

Return

Result of sine for this angle

Example

```
print(sin(0.5)); //will print 0.479426
```

sinh()

```
sinh(x) => Number
```

Calculates hyperbolic sine x

Parameters

x	Number	Angle in radian

Return

Result of hyperbolic sine for this angle

Example

```
print(sinh(0.5)); //will print 0.521095
```

sqrt()

```
sqrt(input) => Float
```

Gets square root from number.

Parameters

input	Number

Return

Square root value.

Example

```
>>> sqrt(4)
2
```

tan()

```
tan(x) => Number
```

Calculates tangent x

Parameters

x	Number	Angle in radian

Return

Result of tangent for this angle

Example

```
print(tan(0.5)); //will print 0.546302
```

tanh()

```
tanh() => void
```

Calculates hyperbolic tangent x

Parameters

x	Number	Angle in radian

Return

Result of hyperbolic tangent for this angle

Example

```
print(tanh(0.5)); //will print 0.462117
```

Time

GetCurrentTimeMs()

```
GetCurrentTimeMs() => Integer64
```

Get current time in milliseconds

Return

Current time in milliseconds

Example

```
>>> GetCurrentTimeMs()
1674586722493
```

gmtime()

```
gmtime(time) => TIME
```

Converts time in UNIX format (number of seconds since epoch) to calendar date and time broken down into its components, expressed as UTC (or GMT timezone). Function uses either time given in time argument or current time if time is omitted.

Parameters

time	Integer	Time as seconds since epoch (1 January 1970 00:00:00 UTC). If omitted, current time is used.

Return

Object of class TIME.

Example

```
println(gmtime(time())->year);   // 2020
println(gmtime()->year);     // 2020
```

localtime()

```
localtime(time) => void
```

Converts time in UNIX format (number of seconds since epoch) to calendar date and time broken down into its components, using server time zone. Function uses either time given in time argument or current time if time is omitted.

Parameters

time	Integer	Time as seconds since epoch (1 January 1970 00:00:00 UTC). If omitted, current time is used.

Return

TIME object.

Example

```
println(localtime(time())->year); //Will print 2020
println(localtime()->year); //Will print 2020
```

mktime()

```
mktime(time) => void
```

Converts broken down time (represented by object of TIME class) to UNIX time (number of seconds since epoch). TIME object can be returned by localtime or gmtime functions or created using operator new. Broken down time assumed to be local time.

Parameters

time	TIME	Broken down time.

Return

UNIX time (number of seconds since epoch).

Example

```
t = new TIME();  // create new TIME object
t->year = 2018;
t->mon = 3;  // April (0-based month numbering)
t->mday = 10;
t->hour = 10;
t->min = 16;
t->sec = 55;
t->isdst = -1;   // auto detect daylight saving
println(mktime(t)); //Will print "1523344615"
```

strftime()

```
strftime(string, time) => String
```

Formats a Unix timestamp, and returns a string according to given formatting rules.

Parameters

string	String	Formatting string - see this for available options http://www.cplusplus.com/reference/ctime/strftime/
time	Integer	UNIX time (number of seconds since epoch).

Return

Formatted time as a string.

Example

```
println(strftime("%Y-%m-%d %H:%M", time())); //Will print: "2016-01-19 12:14"
println(strftime("%Y-%m-%d %H:%M - timezone %Z - offset from UTC - %z", time()));
//Will print: "2016-01-19 12:14 - timezone CET - offset from UTC - +0100"
```

time()

```
time() => void
```

Gets the system time.

Return

System time as number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time, according to the system clock (also known as UNIX time).

Example

```
print(time()); //will print 1588953745
```

Objects

CreateContainer()

This function is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

```
CreateContainer(parent, name) => Container
```

Create new container under `parent` object with desired `name`.

Parameters

parent	Parent object (NetObj referring to container object or infrastructure service root).
name	Name of the container to create

Return

Instance of newly created Container object or `null` if failed.

Example

```

```

CreateNode()

This function is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

```
CreateNode(parent, name, primaryHostName, zoneUIN) => Node
```

Create node object.

Parameters

parent	NetObj	Parent object

name	String	Name for new node
primaryHostName	String	Primary host name for new node (optional)
zoneUIN	Integer	zone UIN (optional)

Return

New node object or null on failure

Example

```
CreateNode(FindObject(2), "SERVER", "10.10.10.1");    // Create node directly under
"Infrastructure Services"
```

FindAccessPointByMACAddress()

```
FindAccessPointByMACAddress(macaddress, currentNode) => AccessPoint
```

Look up AccessPoint object by MAC address of it or BSSID of one of radio interfaces.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

macaddress	String	MAC address.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of AccessPoint object or `null` if not found or not accessible.

Example

```
ap = FindAccessPointByMACAddress("AA:34:33:44:14:55");
if (ap != NULL) println(ap->name); // Prints access point name if it is found
```

FindNodeByAgentId()

```
FindNodeByAgentId(agentid, currentNode) => Node
```

Look up Node object by agent ID.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

agentid	String	Agent ID.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of Node object or `null` if not found or not accessible.

Example

```
node = FindNodeByAgentId("7e54f964-8942-504f-90eb-70e5d6abee92", $node);
if (node != NULL) println(node->name); // Prints node name if node with given agent ID
is found
```

FindNodeByIPAddress()

```
FindNodeByIPAddress(ipaddress, currentNode) => Node
```

Look up Node object by primary IP address or IP address of one of the interfaces.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

ipaddress	String	IP address.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of Node object or `null` if not found or not accessible.

Example

```
node = FindNodeByIPAddress("192.168.1.1", $node);
if (node != NULL) println(node->name); // Prints node name if node with given IP
address is found
```

FindNodeByMACAddress()

```
FindNodeByMACAddress(macaddress, currentNode) => Node
```

Look up Node object by MAC address of one of the interfaces.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

macaddress	String	MAC address.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of Node object or `null` if not found or not accessible.

Example

```
node = FindNodeByMACAddress("AA:34:35:D8:12:45");
if (node != NULL) println(node->name); // Prints node name if node with given MAC
address is found
```

FindNodeBySysName()

```
FindNodeBySysName(sysname, currentNode) => Node
```

Look up Node object by SNMP sysName.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

sysname	String	SNMP sysName.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of Node object or `null` if not found or not accessible.

Example

```
node = FindNodeBySysName("switch-main");
if (node != NULL) println(node->name); // Prints node name if node with given sysName
is found
```

FindNodeObject()

```
FindNodeObject(currentNode, key) => Node
```

Look up Node object by either name or object id, will return `null` if object not found or not accessible. This function search for nodes only.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be set to `null`.

Parameters

currentNode	Lookup context or `null` if trusted nodes validation is disabled.
key	Object name of id.

Return

Instance of Node object or `null` if not found or not accessible.

Example

```
>>> FindNodeObject($node, "server.netxms.org")
object
>>> FindNodeObject(null, 12)
object
>>> FindNodeObject($node, "bad_node_name")
NULL
```

FindObject()

```
FindObject(key, currentNode) => NetObj
```

Look up any object inherited from NetObj (Node, Interface, Cluster, etc.) by either name or object id.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

key	String	Object name or id.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of object inherited from NetObj or `null` if not found or not accessible. Type of the object can be verified using function classof().

Example

```
node = FindObject("Infrastructure Services"); //Will find node with name
"Infrastructure Services"
println(node->name); //Will print "Infrastructure Services"
println(node->id); //Will print "2"
```

FindObjectByGUID()

```
FindObjectByGUID(guid, currentNode) => NetObj
```

Look up any object inherited from NetObj (Node, Interface, Cluster, etc.) by GUID.

If trusted node validation is enforced, `currentNode` should point to execution context object (instance of NetObj, `$node` in most cases). If trusted nodes are disabled (default server configuration), `currentNode` can be omitted or set to `null`.

Parameters

guid	String	Object GUID.
currentNode	NetObj	Lookup context or `null` if trusted nodes validation is disabled. Optional parameter.

Return

Instance of object inherited from NetObj or `null` if not found or not accessible. Type of the object can be verified using function classof().

Example

```
node = FindObjectByGUID("d602a7f1-0049-421b-a134-2fcf35712608", $node);
if (node != NULL) println(node->name); // Prints node name if node with given GUID is
found
```

GetAllNodes()

```
GetAllNodes() => array
```

Get list of all Node objects in the system as array.

Return

Array of node objects.

Example

```
>>> for (n : GetAllNodes()) {
>>>   println(n->id . " - " . n->name);
```

```
>>> }

6766 - demo-netxms
6901 - Control Unit 1
6902 - Control Unit 2
```

GetServerNode()

```
GetServerNode() => Node
```

Get node object of the management server.

Return

Node instance.

Example

```
>>> GetServerNode()
Node@0x7facac2889e0
```

GetServerNodeId()

`GetServerNodeId() ⇒ Integer`

Get ID of management server's node.

Return

Object ID of the current server node.

Example

```
>>> GetServerNodeId()
161
```

Data collection

CreateDCI()

```
CreateDCI(node, origin, name, description, dataType, pollingInterval, retentionTime)
=> DCI
```

Create new data collection item on node, return DCI object instance of `null` if failed.

Parameters

node	Node object instance (e.g. `$node`), where DCI should be created.

origin	Data origin. Possible values: "agent", "snmp", "internal", "push", "websvc", "winperf", "script", "ssh", "mqtt", "driver", "cpsnmp". Alternatively, you can use Data origin constants.
name	name of the metric (e.g. "Agent.Version")
description	human readable description.
dataType	Type of the collected data. Possible values: "int32", "uint32", "int64", "uint64", "counter32", "counter64", "float", "string". Alternatively, you can use DCI data type constants.
pollingInterval	polling interval in seconds or `null` for server-default value.
retentionTime	retention time in days, `0` for "Do not save collected data to database" or `null` for server-default value.

Return

Instance of newly created DCI of `null` if failed.

Example

```
>>> d = CreateDCI($node, DataSource::AGENT, "Agent.Version", "Agent Version",
DCI::STRING, 0, 0);
>>> println(d->id);
145
```

FindAllDCIs()

```
FindAllDCIs(node, nameFilter, descriptionFilter) => Array
```

Find all DCI on the `node` matching `nameFilter` **and** `descriptionFilter`. Filter can contain glob symbols "?" and "*". If filter is `null`, it's ignored.

Parameters

node	Node object instance (e.g. `$node`)
nameFilter	GLOB for mastching DCI name or `null` if name should be ignored.
descriptionFilter	GLOB for mastching DCI description or `null` if description should be ignored.

Return

Array of DCI.

Example

```
>>> list = FindAllDCIs($node, "Server*", "*MAIN*");
>>> foreach (row : list) {
>>>     println(row->id . ": " . row->description . " (" . row->name . ")");
>>> }
```

```
91: Server thread pool MAIN: usage (Server.ThreadPool.Usage(MAIN))
92: Server thread pool MAIN: normalized load average (1 minute) (Server.ThreadPool
.LoadAverage(MAIN,1))
93: Server thread pool MAIN: current load (Server.ThreadPool.Load(MAIN))

>>> list = FindAllDCIs($node, "Server*");
>>> foreach (row : list) {
>>>     println(row->id . ": " . row->description . " (" . row->name . ")");
>>> }
100: NetXMS server: database writer's request queue (other queries) (Server
.AverageDBWriterQueueSize.Other)
101: NetXMS server: database writer's request queue (Server.AverageDBWriterQueueSize)
103: NetXMS server: data collector's request queue (Server
.AverageDataCollectorQueueSize)
...

>>> list = FindAllDCIs($node, null, "*load average*");
>>> foreach (row : list) {
>>>     println(row->id . ": " . row->description . " (" . row->name . ")");
>>> }
119: CPU: load average (15 minutes) (System.CPU.LoadAvg15)
123: CPU: load average (5 minutes) (System.CPU.LoadAvg5)
83: Server thread pool AGENT: normalized load average (1 minute) (Server.ThreadPool
.LoadAverage(AGENT,1))
...
```

FindDCIByDescription()

```
FindDCIByDescription(node, description) => Integer
```

Find ID of the DCI on node by description (exact match). FindAllDCIs() can be used for pattern search.

Parameters

node	Node object instance (e.g. $node)
description	Description of the DCI

Return

Integer ID of the DCI or 0 if not found.

Example

```
>>> d = FindDCIByDescription($node, "Agent Version");
>>> print(d);
144
```

FindDCIByName()

```
FindDCIByName(node, dciParameter) => Integer
```

Find ID of the DCI on node by parameter name (exact match). FindAllDCIs() can be used for pattern search.

Parameters

node	Node object instance (e.g. $node)
dciParameter	Parameter name of the DCI

Return

Integer ID of the DCI or 0 if not found.

Example

```
>>> d = FindDCIByName($node, "Agent.Version");
>>> print(d);
144
```

GetAvgDCIValue()

```
GetAvgDCIValue(object, dciId, periodStart, periodEnd) => Number
```

Get the average value of the DCI for the given period. The DCI value type must be numeric.

Parameters

object	Instance of Node, Cluster, or MobileDevice object (e.g. $node).
dciId	ID of the DCI to retrive.
periodStart	Unix timestamp of the period start.
periodEnd	Unix timestamp of the period end.

Return

Average value or null on failure.

Example

```
>>> obj = FindObject("Server1");
>>> dciID = FindDCIByName(obj, "CPU.Usage")
>>> val = GetAvgDCIValue(obj, dciId, 0, time()); // time range from January 1, 1970
untill now
>>> println("Average CPU Usage: ". val . "%");
Average CPU Usage: 17%
```

GetDCIObject()

```
GetDCIObject(node,id) => DCI
```

Get DCI object by id

Parameters

node	Node	Node object instance
id	Number	DCI id

Return

DCI object or `null` if not found

Example

```
GetDCIObject($node, 2)      // object
GetDCIObject($node, bad_id) // `null`
```

GetDCIRawValue()

```
GetDCIRawValue(node,id) => Last Value
```

Returns last raw value of DCI

Parameters

node	Node	Node object instance (e.g.)
id	Integer	DCI ID

Return

Last raw value (before transformation) for given DCI or null if DCI with given ID does not exist or has no collected values.

Example

```
GetDCIRawValue($node, FindDCIByName($node, "Status")) // 0
```

GetDCIValue()

```
GetDCIValue(node, dciId) => String or Table
```

Get last collected value of the DCI. Return `null` on error, Table instance for table DCI or String otherwise.

Parameters

node	Node	node object instance (e.g. `$node`)
dciId	Integer	DCI ID

Return

Table for table DCIs, String, or `null` if failed or no data is available.

Example

```
>>> GetDCIValue($node, FindDCIByName($node, "Status"))
0
>>> GetDCIValue($node, FindDCIByName($node, "Non-Existing"))
null
```

GetDCIValueByDescription()

```
GetDCIValueByDescription(node, description) => String or table
```

Get last value of DCI with given description on given node.

Parameters

node	Node	Node object instance (e.g.)
description	String	DCI description.

Return

Last value for given DCI (String for normal DCIs and Table object for table DCIs) or null if DCI with given description does not exist or has no collected values.

Example

```
GetDCIValueByDescription($node, "Status")   // 0
```

GetDCIValueByName()

```
GetDCIValueByName(node, name) => String or Table
```

Get last value of DCI with given name on given node.

Parameters

node	Node	Node object instance (e.g.)
name	String	DCI name (parameter's name for agent or internal source, and OID for SNMP source).

Return

Last value for given DCI (string for normal DCIs and Table object for table DCIs) or null if DCI with given name does not exist or has no collected values.

Example

```
GetDCIValueByName($node, "Agent.Version")   // "1.2.0"
```

GetDCIValues()

```
GetDCIValues(node, id, startTime, endTime, rawValues) => Array
```

Get all values for period of DCI with given ID on given node.

Parameters

node	Node	node object instance
id	Integer	DCI ID
startTime	Integer	Start of the period (as UNIX timestamp).
endTime	Integer	End of the period (as UNIX timestamp). Optional, current time will be used if this parameter is not specified
rawValues	Boolean	False - return transformed values. True - return RAW values. Optional, False by default. **This parameter was added in version 4.4.1.**

Return

Array of value ordered from latest to earliest for given DCI or null if DCI with given ID does not exist or has no collected values. This function cannot be used for table DCIs.

Example

```
GetDCIValues($node, FindDCIByName($node, "Status"), time() - 3600, time()); // Values
for last hour
```

GetMaxDCIValue()

```
GetMaxDCIValue(node, dciId, from, to) => String
```

Get the maximum value of the DCI for the given period. The DCI value must be of numeric type.

Parameters

node	Node	node object instance (e.g.)
id	Integer	DCI ID

from	Integer	Start of the period (as UNIX timestamp).
to	Integer	End of the period (as UNIX timestamp).

Return

Maximum value or null on failure.

Example

```
value = GetMaxDCIValue(FindObject("MYWORKPC"), 18, 0, time()); //Max value from the
beginning till now\
println(value); //Will print maximum value
```

GetMinDCIValue()

```
GetMinDCIValue(node, dciId, from, to)) => String
```

Get the minimum value of the DCI for the given period. The DCI value must be of numeric type.

Parameters

node	Node	node object instance (e.g.)
id	Integer	DCI ID
from	Integer	Start of the period (as UNIX timestamp).
to	Integer	End of the period (as UNIX timestamp).

Return

Minimum value or null on failure.

Example

```
value = GetMinDCIValue(FindObject("MYWORKPC"), 18, 0, time()); //Minimal value from
the beginning till now
println(value); //Will print minimal value
```

GetSumDCIValue()

```
GetSumDCIValue(node, dciId, from, to) => String
```

Get the sum value of the DCI for the given period. The DCI value must be of numeric type.

Parameters

node	Node	node object instance (e.g.)
id	Integer	DCI ID

from	Integer	Start of the period (as UNIX timestamp).
to	Integer	End of the period (as UNIX timestamp).

Return

Sum value or null on failure.

Example

```
value = GetSumDCIValue(FindObject("MYWORKPC"), 18, 0, time()); //sum value from the
beginning till now
println(value); //Prints value
```

PushDCIData()

```
PushDCIData(node, dciId, value) => void
```

Push new DCI value from script.

Parameters

node	Node	node object instance
dciId	Integer	DCI id for which new value will be pushed (DCI source must be set to "Push").
value	Integer or String	New value for DCI.

Return

No return value

Example

```
PushDCIData($node, 46, 13); //Will push value "13" to DCI with id 46
```

Alarms

FindAlarmById()

```
FindAlarmById(id) => Alarm
```

Finds alarm object by id

Parameters

id	Integer	Alarm id

Return

Alarm object or null if no such alarm exist.

Example

```
alarm = FindAlarmById(72388); // Will find alarm with id 72388
println(alarm->id); // Will print "72388"
println(alarm->message); // Will print alarm message. For example: "Node down"
```

FindAlarmByKey()

```
FindAlarmByKey(key) => Alarm
```

Find alarm by key

Parameters

key	String	Alarm key

Return

Will return object of class alarm

Example

```
alarm = FindAlarmByKey("NODE_DOWN_0x000023A2"); //Will find alarm with key
"NODE_DOWN_0x000023A2"
println(alarm->key); //Will print key "NODE_DOWN_0x000023A2"
println(alarm->message); //Will print alarm message, for example "Node down"
```

FindAlarmByKeyRegex()

```
FindAlarmByKeyRegex(key) => Alarm
```

Find alarm by key using regular expression

Parameters

key	String	Key regular expression

Return

Alarm object found by key

Example

```
alarm = FindAlarmByKeyRegex("NODE_DOWN_.*"); //Will find alarm with regexp key
"NODE_DOWN_.*"
println(alarm->key); //Will print key "NODE_DOWN_0x00001628"
println(alarm->message); //Will print alarm message, for example "Node down"
```

Events

LoadEvent()

```
LoadEvent(eventId) => Event
```

Will load event from the database and return Event object or NULL if event not found.

Parameters

eventId	Id of the event to be loaded from database

Return

Event loaded from database or NULL if event not found.

Example

```
>>>event = LoadEvent(315);
>>>event->id;
315
```

PostEvent()

```
PostEvent(node, event, tag, ...) => Boolean
```

Post event on behalf of given node.

Parameters

node	Node	Node object to send event on behalf of.
event	Integer or String	Event code or name (name can be used since 1.2.6).
tag	String	User tag associated with event. Optional, can be leaved out or set to null.
...	String	0 or more event-specific parameters.

Return

TRUE if event was posted successfully or FALSE if not.

Example

```
PostEvent($node, 100000);
PostEvent($node, 100000, "my tag", "param1", "param2");
PostEvent($node, "MY_EVENT_NAME", null, "param1");
```

PostEventEx()

```
PostEventEx(node, event, parameters, tags, originTimestamp, origin) => Boolean
```

Post event on behalf of given node.

Parameters

node	Node	Node object to send event on behalf of.
event	Integer or String	Event code or name.
parameters	Array or Hash map	Event parameters. Hash map allows to use named parameters - hash map keys will become parameter names. Optional, can be set to null.
tags	Array of Strings	User tags associated with event. Optional, can be set to null.
originTimestamp	Integer	Time as seconds since epoch (1 January 1970 00:00:00 UTC). Optional, can be set to null.
origin	Integer	One of the possible values. Optional, can be set to null.

Return

`TRUE` if event was posted successfully or `FALSE` if not.

Example

```
// event parameters as array
r = PostEventEx($node, "MY_EVENT", %("text parameter", 123));

// event parameters as hash map
r = PostEventEx($node, "MY_EVENT", %{"someParam":"value", "anotherParam":456});

// null is used to omit originTimestamp
params = %{};
params["someParam"] = "some value";
params["anotherParam"] = 42;
tags = %();
tags->append("tag1");
tags->append("tag2");
r = PostEventEx($node, "MY_EVENT", params, tags, null, EventOrigin::NXSL);
```

Miscelanious

_exit()

`_exit(exitCode=0) ⇒ void`

Stop script execution and return `exitCode`.

Parameters

exitCode	Integer	Optional exit code for the script. Defaults to `0`.

assert()

```
assert(expression) => void
```

Check if given value is `TRUE`

Parameters

expression	Boolean	Expression or value to evaluate

Example

```
node = FindObject("aaa");
assert(node != null); //Will go through, if object with name "aaa" found and will exit
script if not
```

classof()

```
classof(instance) => String
```

Return class name of the `instance`.

Parameters

instance	Instance of any class.

Return

Class name.

Example

```
>>> classof($node)
Node
```

CountryAlphaCode()

```
CountryAlphaCode(code) => String
```

Lookup country alpha code by numeric or alpha3 code.

Parameters

code	Numeric (3 digits) or 3-letter country code.

Return

Two letter country code or `null` if country not found.

Example

```
>>> CountryAlphaCode('020')
AN
>>> CountryAlphaCode('AND')
AN
>>> CountryAlphaCode('124')
CA
```

CountryName()

```
CountryName(code) => String
```

Get country name from code

Parameters

code	String	Country code

Return

Country name

Example

```
print(CountryName("BE")); // Will print "Belgium"
```

CountScheduledTasksByKey()

```
CountScheduledTasksByKey(key) => Integer
```

Will count scheduled tasks by key

Parameters

key	String	Schedule key

Return

Scheduled task count with provided key

Example

```
print(CountScheduledTasksByKey("Key")); //Number of tasks
```

CreateUserAgentNotification()

```
CreateUserAgentNotification(object, message, startTime, endTime) => Number
```

Creates user agent notification

Parameters

object	Node or root object to send notification
message	Message to be sent to clients
startTime	Start time of notification delivery
endTime	End time of notification delivery

Return

New user agent notification id

Example

```
>>> CreateUserAgentNotification($node, "One time notification text", 0, 0);
14

>>> CreateUserAgentNotification($node, "Interval user agent notification text",
time(), time()+86400);
15
```

CurrencyAlphaCode()

```
CurrencyAlphaCode(code) => String
```

Get currency alpha code from numeric code

Parameters

code	String	Numeric code as a string

Return

Currency alpha code from numeric code

Example

```
print(CurrencyAlphaCode("978")); // Will print "EUR"
```

CurrencyExponent()

```
CurrencyExponent(code) => Integer
```

Get currency exponent

Parameters

code	String	Currency numeric or character code as a String

Return

Currency exponent

Example

```
println(CurrencyExponent("EUR")); // Will print 2
println(CurrencyExponent("978")); //Will print 2
```

CurrencyName()

```
CurrencyName(code) => String
```

Get currency name from code

Parameters

code	String	Currency numeric or character code as a String

Return

Currency name

Example

```
println(CurrencyName("EUR")); //Will print "Euro"
println(CurrencyName("978")); //Will print "Euro"
```

FormatMetricPrefix()

```
FormatMetricPrefix(number, binary, precision) => String
```

Convert numeric value to human-readable form with metric prefix.

Parameters

number	Number	Number to format.
binary	Boolean	Optional argument. Set to true to use binary multipliers (Ki, Mi, etc.). Default value is false.
precision	Unsigned Integer	Optional argument with rounding precision. Acceptable values are from 0 to 20. Default value is 2.

Return

String containing the value converted to human-readable form with metric prefix.

Example

```
>>> FormatMetricPrefix(5000)
5.00k
>>> FormatMetricPrefix(5000, true)
4.88Ki
>>> FormatMetricPrefix(-1234567.89, false, 0)
-1M
>>> FormatMetricPrefix(-1234567.89,, 10)
-1.2345678900M
```

FindVendorByMACAddress()

```
FindVendorByMACAddress(text) => String
```

Find NIC vendor information by mac address

Parameters

text	String	MAC address

Return

NIC vendor name or null

Example

```
>>> FindVendorByMACAddress("00:00:5e:00:53:af")
ICANN, IANA Department
```

GetConfigurationVariable()

```
GetConfigurationVariable(key, defaultValue=null) => String
```

Read server configuration parameter by `key`.

Parameters

key	Configuration parameter name to lookup.
defaultValue	Optional argument with default value if key is not found.

Return

Value of the server configuration parameter. If key is not found, `null` is returned or `defaultValue` is specified.

Example

```
>>> GetConfigurationVariable("NumberOfStatusPollers")
10
>>> GetConfigurationVariable("BadVariable")
NULL
>>> GetConfigurationVariable("BadVariable", 22)
22
```

gethostbyaddr()

```
gethostbyaddr(address) => String
```

Resolve IP address to host name

Parameters

address	String	IP address to resolve host name

Return

Host name

Example

```
println(gethostbyaddr("127.0.0.1")); //Will print "localhost"
```

gethostbyname()

```
gethostbyname(hostname) => String
```

Resolve hostname to IP address

Parameters

hostname	String	Hostname

Return

IP address as a String

Example

```
println(gethostbyname("localhost")); //Will print "127.0.0.1"
```

GetMappingTableKeys()

```
GetMappingTableKeys(table) => Array
```

Get array with all keys from mapping table.

Parameters

table	Table name or ID

Return

Array of keys or `null` if mapping table is not found.

Example

Provided that there is a mapping table named "table1", which has keys 10, 20 and 30 with corresponding values "value1", "value2" and "value3":

```
println(GetMappingTableKeys("table1")); // prints "[10, 20, 30]"
```

GetServerQueueNames()

```
GetServerQueueNames() => Array
```

Get array with server queue names.

Return

Array with strings.

Example

```
foreach(l : GetServerQueueNames())
{
  println(l);
}
```

GetSyslogRuleCheckCount()

```
GetSyslogRuleCheckCount(name, object) => Integer
```

Get syslog rule check count for all objects or for exact object.

Parameters

name	String	Rule name
object	NetObj or Integer	Object of class NetObj or object id. Optional parameter

Return

Syslog rule check count or -1 if syslog daemon not initialized

Example

```
println(GetSyslogRuleCheckCount("RuleName", $object)); // Will print rule check count
for rule with name "RuleName"
```

GetSyslogRuleMatchCount()

```
GetSyslogRuleMatchCount(name, object) => Integer
```

Get syslog rule match count or all objects or for exact object.

Parameters

name	String	Rule name
object	NetObj or Integer	Object of class NetObj or object id. Optional parameter

Return

Syslog rule match count or -1 if syslog daemon not initialized

Example

```
println(GetSyslogRuleMatchCount("RuleName", $object)); // Will print rule match count
for rule with name "RuleName"
```

GetThreadPoolNames()

```
GetThreadPoolNames() => Array
```

Get array with thread pool names.

Return

Array with strings.

Example

```
print(GetThreadPoolNames()[0]); //will print "MAIN"
```

JsonParse()

```
JsonParse(string) => JSON object
```

Parse input `string` to JsonObject

Parameters

string	JSON as a string.

Return

JsonObject if root object is Json object and JsonArray if root object is Json array or null if failed to parse

map()

```
map(table, key, default=null) => String
```

Lookup value from mapping table.

Parameters

table	Table name or ID.
key	String key to lookup.
default	Optional default value.

Return

When key or table is not found, return `null` or default value if provided.

Example

Provided that there is a mapping table named "table1", which has keys 10, 20 and 30 with corresponding values "value1", "value2" and "value3":

```
println(map("table1", 20)); // prints "value2"
println(typeof(map("table1", 40))); // prints "null"
println(map("table1", 40, "NOT FOUND")); // prints "NOT FOUND"
```

mapList()

```
mapList(table, list, separator, default) => String
```

Lookup multiple keys (separated by user-defined separator) from mapping table. Result string is joined using the same separator.

Parameters

table	name or ID of the mapping table.
list	string of keys separated by `separator`.
separator	separator used to split `list` and to produce output.

Example

Provided that there is a mapping table named "table1", which has keys 10, 20 and 30 with corresponding values "value1", "value2" and "value3":

```
println(mapList("table1", "10,20,30", ",")); // prints "value1,value2,value3"
println(mapList("t", "10,20,30,40", ",", "NOT FOUND"));  // prints
"value1,value2,value3,NOT FOUND"
```

print()

```
print(object) => void
```

Prints any text or text representation of object

Parameters

object	object	Any object

Example

```
a = %("a", 1, 1.5);
print("Array: ");
print(a); //Result: Array: [a, 1, 1.500000]
```

println()

```
println(object) => void
```

Prints any text or text representation of object and adds new line symbol after it

Parameters

object	object	Any object

Example

```
a = %("a", 1, 1.5);
println("Array: ");
println(a);
//Array:
//[a, 1, 1.500000]
```

random()

```
random(minValue, maxValue) => Integer
```

Generate pseudo random number in given range. Uses c/c++ rand() function.

Parameters

minValue	Integer	Start of range.
minValue	Integer	End of range.

Return

Random value in range minValue..maxValue.

Example

```
println(random(0, 100)); // Will print random value in 0..100 range
```

ReadPersistentStorage()

```
ReadPersistentStorage(key) => String
```

Read value from global persistent key-value storage.

Parameters

key	String record key to lookup.

Return

Value referenced by `key` or `null` if key not exist.

Example

```
>>> ReadPersistentStorage("key1")
value1
>>> ReadPersistentStorage("key2")
null
>>> WritePersistentStorage("key2", "value2")
>>> ReadPersistentStorage("key2")
value2
```

SecondsToUptime()

```
SecondsToUptime(seconds) => String
```

Format system uptime in seconds as string in format "n days, hh:mm".

Parameters

seconds	Integer	Number of seconds.

Return

System uptime in format "n days, hh:mm".

Example

```
println(SecondsToUptime(600)); // Will print "0 days, 00:10"
```

SendMail()

```
SendMail(recipients, subject, text, ishtml) => void
```

Sends e-mail to specified recipients.

Parameters

recipients	String	List of e-mail addresses. Multiple recipients should be separated by ";".
subject	String	E-mail subject.
text	String	E-mail text (body).
ishtml	Boolean	Indicates whether the text is in HTML. If omitted then false is assumed.

Return

No return value

Example

```
SendMail("somebody@somewhere.com", "My subject", "My text", false);
```

SendNotification()

```
SendNotification(channelName, recipients, subject, text) => void
```

Sends notification to specified recipients.

Parameters

channelName	String	Name of notification channel.
recipients	String	List of recipients. Multiple recipients should be separated by ";".
subject	String	Subject. Not all notification channel may support this field, in this case empty string "" should be provided.
text	String	Message text.

Return

No return value

Example

```
SendNotification("Telegram", "Alice; Bob", "", "My message text");
```

sleep()

```
sleep(milliseconds) => void
```

Suspend script execution for given number of milliseconds.

Parameters

milliseconds	Integer	Number of milliseconds to suspend script execution for.

Return

No return value

Example

```
sleep(1000);   // sleep for 1 second
```

trace()

```
trace(debugLevel, message) => void
```

Writes `message` to NetXMS main log if current debug level is equal or higher than `debugLevel`.

Parameters

debugLevel	Target debug level.
message	String to be written.

Example

```
>>> trace(0, "Test");
```

typeof()

```
typeof(instance) => string
```

Return type of the `instance`.

Parameters

instance	Instance of the object or primitive.

Return

Name of the type.

Example

```
>>> typeof(1)
int32
>>> typeof(1L)
int64
>>> typeof(%(1, 2, 3))
array
>>> typeof(new Table())
object
>>> typeof(null)
null
```

PollerTrace()

```
PollerTrace(message) => void
```

Sends poller trace message to client (if poll initiated by client) and log it. Used only in hooks.

Parameters

message	String	Message text

WritePersistentStorage()

```
WritePersistentStorage(key, value) => void
```

Create or update value in global persistent key-value store.

Parameters

key	String key.
value	String value to be saved.

Example

```
>>> WritePersistentStorage("key1", "value1")
>>> ReadPersistentStorage("key1")
value1
```

weierstrass()

```
weierstrass(a, b, x) => Number
```

Calculate Weierstrass function for given x, a, and b. More can be found there: https://en.wikipedia.org/wiki/Weierstrass_function Can be used for test data generation.

Parameters

a	Number	Coefficient, 0<a<1
b	Number	Coefficient, odd integer
x	Number	Point to calculate function in

Return

Value in point x for Weierstrass function with given coefficients

Example

```
print(weierstrass(0.5, 7, 10)); //will print "1.999218"
```

Hashes and encoding

Base64Decode()

```
Base64Decode(string,encoding) => String
```

Decode base64 encoded string. Accepts string to encode and optional character encoding. Valid character encodings are "UTF-8", "UCS-2", "UCS-4", "SYSTEM". Default is UTF-8.

Parameters

string	String	String to decode
encoding	String	String encoding. Optional parameter.

Return

Decoded string

Example

```
print(Base64Decode("U29tZSB0ZXh0")); //Will print "Some text"
```

Base64Encode()

```
Base64Encode() => String
```

Encode string as base64. Accepts string to encode and optional character encoding. Valid character encodings are "UTF-8", "UCS-2", "UCS-4", "SYSTEM". Default is UTF-8.

Parameters

string	String	String to encode
encoding	String	String encoding. Optional parameter.

Return

Encoded string

Example

```
print(Base64Encode("Some text")); //Will print "U29tZSB0ZXh0"
```

md5()

```
md5(string) => String
```

The MD5 message-digest algorithm implementation

Parameters

string	String	String to get hash

Return

MD5 hash

Example

```
println(md5("Some text")); //Will print "9DB5682A4D778CA2CB79580BDB67083F"
```

sha1()

```
sha1(string) => String
```

sha1 function implementation

Parameters

string	String	String to get result of Secure Hash Algorithm 1

Return

Result of sha1

Example

```
println(sha1("String")); //Will pritn "3DF63B7ACB0522DA685DAD5FE84B81FDD7B25264"
```

sha256()

```
sha256(string) => String
```

sha256 function implementation

Parameters

string	String	String to get result of Secure Hash Algorithm 256

Return

Result of sha256

Example

```
println(sha1("String")); //Will pritn
"B2EF230E7F4F315A28CDCC863028DA31F7110F3209FEB76E76FED0F37B3D8580"
```

Files

Available only if 'NXSL.EnableFileIOFunctions' configuration parameter is turned on.

CopyFile()

```
CopyFile(sourceFileName, destinationFileName) => Boolean
```

Makes a copy of source file or folder to destination file or folder and returns `TRUE` if action was successful.

Parameters

sourceFileName	String	Source file/folder name
destinationFileName	String	Destination file/folder name

Return

`TRUE` if copy was successful and `FALSE` if not.

Example

```
CopyFile("/tmp/source", "/tmp/destination") // Will copy "/tmp/source" file content to
```

```
"/tmp/destination" file.
```

CreateDirectory()

```
CreateDirectory(newDirectory) => Boolean
```

Makes a copy of source file to destination file and returns `TRUE` if action was successful. Will create destination file if file does not exist.

Parameters

newDirectory	String	Directory to be created

Return

`TRUE` if creation was successful and `FALSE` if not.

Example

```
CreateDirectory("/tmp/newDir") // Will create new directory with name "newDir" under
folder "tmp"
```

DeleteFile()

```
DeleteFile(file) => Boolean
```

Deletes file.

Parameters

file	String	File to delete

Return

`TRUE` if delete was successful and `FALSE` if not.

Example

```
DeleteFile("/tmp/file") // Will delete "/tmp/file" file.
```

FileAccess()

```
FileAccess(fileName, mode) => Boolean
```

Check file access.

Parameters

fileName	String	File to check access on
mode	Integer	Mode to check

Windows access mode

00	Existence only
02	Write-only
04	Read-only
06	Read and write

Unix like system access mode

0	Existence only
1	Execute permission
2	Write permission
4	Read permission

Return

`TRUE` if user has given access to provided file and `FALSE` if not.

Example

```
return FileAccess("/tmp/file", 5) // Will return true if user has read and execute
access to file
```

OpenFile()

```
OpenFile(fileName, fileMode) => File object
```

Open file. Returns FILE object or null.

Parameters

fileName	String	File name to open.
openingMode	String	Mode file should be opened in. More can be found there.

Return

File class or `NULL` if error occurred.

Example

```
OpenFile("/tmp/source", "/tmp/destination") // Will copy "/tmp/source" file content to
"/tmp/destination" file.
```

RemoveDirectory()

```
RemoveDirectory(directory) => Boolean
```

Removes directory.

Parameters

directory	String	Directory to delete

Return

`TRUE` if delete was successful and `FALSE` if not.

Example

```
RemoveDirectory("/tmp/dir") // Will delete "/tmp/dir" directory.
```

RenameFile()

```
RenameFile(currentFile, newFileName) => Boolean
```

Renames file.

Parameters

currentFile	String	File to rename
newFileName	String	New file name

Return

`TRUE` if renamed was successful and `FALSE` if not.

Example

```
RenameFile("/tmp/oldFileName", "/tmp/mewFileName") // File with name "oldFileName"
will be renamed to "mewFileName"
```

Class Reference

AccessPoint

Represents NetXMS access point object.

Instance attributes

`icmpAverageRTT ⇒ Integer`

ICMP average response time for primary address. Will return null if no information.

`icmpLastRTT ⇒ Integer`

ICMP last response time for primary address. Will return null if no information.

`icmpMaxRTT ⇒ Integer`

ICMP maximal response time for primary address. Will return null if no information.

`icmpMinRTT ⇒ Integer`

ICMP minimal response time for primary address. Will return null if no information.

`icmpPacketLoss ⇒ Integer`

ICMP packet loss for primary address. Will return null if no information.

`index ⇒ Integer`

Index

`model ⇒ String`

Model

`node ⇒ Node`

Parent node

`serialNumber ⇒ String`

Serial number

`apState ⇒ String`

State from Access point state

`vendor ⇒ String`

Vendor

Constants

Access point state

Description	Value
AP_UP	0

Description	Value
AP_UNPROVISIONED	1
AP_DOWN	2
AP_UNKNOWN	3

Alarm

Represents NetXMS alarm.

Instance attributes

`ackBy ⇒ Number`

ID of user who acknowledged this alarm.

`categories ⇒ Array`

Array with alarm category names.

`creationTime ⇒ Number`

Unix timestamp of the alarm creation time.

`dciId ⇒ Number`

If alarm was created as a result of DCI threshold violation, this attribute will contain ID of the DCI.

`eventCode ⇒ Number`

Event code of originating event.

`eventId ⇒ Number`

ID of originating event.

`eventName ⇒ String`

Name of originating event.

`eventTagList ⇒ Number`

List of event tags as a coma separated string

`helpdeskReference ⇒ String`

Helpdesk system reference (e.g. issue ID).

`helpdeskState ⇒ Number`

Helpdesk state:

- 0 = Ignored
- 1 = Open
- 2 = Closed

`id ⇒ Number`

Unique identifier of the alarm.

`impact ⇒ String`

Alarm impact text

`key ⇒ String`

Alarm key.

`lastChangeTime ⇒ Number`

Unix timestamp of the last update.

`message ⇒ String`

Alarm message.

`originalSeverity ⇒ Number`

Original severity of the alarm.

`parentId ⇒ Number`

Parent alarm id

`repeatCount ⇒ Number`

Repeat count.

`resolvedBy ⇒ Number`

ID of user who resolved this alarm.

`rcaScriptName ⇒ String`

Name of root cause analysis script

`ruleGuid ⇒ String`

Guid of the rule that generated the event.

`ruleDescription ⇒ String`

Description of the rule that generated the event.

`severity ⇒ Number`

Current alarm severity.

`sourceObject ⇒ Number`

ID of the object where alarm is raised.

`state ⇒ Number`

Alarm state:

- 0 = Outstanding
- 1 = Acknowledged
- 2 = Resolved

- 17 = Sticky acknowledged

Instance methods

`acknowledge() ⇒ Number`

Acknowledge alarm. Return `0` on success or error code on failure.

`expandString(string) ⇒ String`

Expand string by replacing macros with their values.

Parameters

string	String to expand

Return

Formated string

`resolve() ⇒ Number`

Resolve alarm. Return `0` on success or error code on failure.

`terminate() ⇒ Number`

Terminate alarm. Return `0` on success or error code on failure.

`addComment(commentText, syncWithHelpdesk) ⇒ Number`

Add new alarm comment.

Parameters

commentText	String	Text of the new alarm comment.
syncWithHelpdesk	String	Optional. If synchronization with helpdesk should be done. TRUE by default.

Return

Id of the newly created alarm comment.

`getComments() ⇒ Array`

Get array of alarm comments.

Return

Array of AlarmComment objects.

AlarmComment

Represents NetXMS alarm comment.

Instance attributes

`id ⇒ Number`

Alarm comment ID.

`changeTime ⇒ Number`

Unix timestamp of the alarm comment last modification time.

`userId ⇒ Number`

ID of user who last modified this alarm commant.

`text ⇒ String`

Alarm comment text.

Asset

Object represent asset, extends NetObj

Instance attributes

`linkedObject ⇒ NetObj`

Linked object

`linkedObjectId ⇒ Integer`

ID of the linked object

`properties ⇒ Asset Properties`

Reference to asset properties class.

Asset Properties

Object contains all asset properties accessible by property name. So value of asset property with name ipAddr can be accesed like:

```
$node->assetProperties->ipAddr
```

BusinessService

Object represents business service

Instance attributes

`checks ⇒ Array`

List of checks an array of BusinessServiceCheck class objects

`instance ⇒ String`

Instance of the check

`prototypeId ⇒ Integer`

ID of the prototype object

`serviceState ⇒ Integer`

Current state of the business service one of states

BusinessServiceCheck

Object represents business service check

Instance attributes

`currentTicketId ⇒ Integer`

ID of the ticket that is currently associated with this check or 0 if check is OK

`description ⇒ String`

Cehck description

`failureReason ⇒ String`

Reason why check have failed or null if check is not failed

`id ⇒ Integer`

ID of the check

`relatedDCI ⇒ DCI`

DCI object of class DCI that is checked or NULL if check is not DCI based

`relatedDCIId ⇒ Integer`

DCI id that is checked or 0 if it is not DCI based check

`relatedObject ⇒ NetObj`

Related object (inherited from NetObj, exact type can be checked with classof() function) or NULL

`relatedObjectId ⇒ Integer`

ID of related objec or 0

`state ⇒ Integer`

Current state of the business service check one of states

`type ⇒ Integer`

Business service check one of types

ByteStream

Object represents stream of bytes

Instance attributes

`pos ⇒ Integer`

Current position in the byte stream

`size ⇒ Integer`

Size of byte stream in bytes

`eos ⇒ boolean`

Indicates, if current position is in the end of byte stream

Instance methods

`seek(position) ⇒ Integer`

Change current position in byte stream.

Parameters

position	Integer	Position to seek to

Return

Actual position if position change was successful or `-1` if position change was not successful.

`readByte() ⇒ Integer (32 bit)`

Read byte from byte stream and return it's value.

`readInt16B() ⇒ Integer (32 bit)`

Read big-endian 16-bit signed integer from byte stream and return it's value.

`readInt32B() ⇒ Integer (32 bit)`

Read big-endian 32-bit signed integer from byte stream and return it's value.

`readInt64B() ⇒ Integer (64 bit)`

Read big-endian 64-bit signed integer from byte stream and return it's value.

`readUInt16B() ⇒ Integer (32 bit)`

Read big-endian 16-bit unsigned integer from byte stream and return it's value.

`readUInt32B() ⇒ Integer (32 bit)`

Read big-endian 32-bit unsigned integer from byte stream and return it's value.

`readUInt64B() ⇒ Integer (64 bit)`

Read big-endian 64-bit unsigned integer from byte stream and return it's value.

`readFloatB() ⇒ Floating-point number`

Read 8 bytes from stream as big-endian floating-point number and return it's value.

`readInt16L() ⇒ Integer (32 bit)`

Read little-endian 16-bit signed integer from byte stream and return it's value.

`readInt32L() ⇒ Integer (32 bit)`

Read little-endian 32-bit signed integer from byte stream and return it's value.

`readInt64L() ⇒ Integer (64 bit)`

Read little-endian 64-bit signed integer from byte stream and return it's value.

`readUInt16L() ⇒ Integer (32 bit)`

Read littleig-endian 16-bit unsigned integer from byte stream and return it's value.

`readUInt32L() ⇒ Integer (32 bit)`

Read little-endian 32-bit unsigned integer from byte stream and return it's value.

`readUInt64L() ⇒ Integer (64 bit)`

Read little-endian 64-bit unsigned integer from byte stream and return it's value.

`readFloatL() ⇒ Floating-point number`

Read 8 bytes from stream as little-endian floating-point number and return it's value.

`readCString() ⇒ String`

Read C-string from stream and return it's value. It's expected that string in the stream is terminates by a null character.

`readPString() ⇒ String`

Read Pascal string from stream and return it's value. It's expected that first byte of string in the stream contains length of the string.

`readString(length) ⇒ String`

Read string from stream and return it's value.

Parameters

length	Integer	Length of the string

`writeByte(value) ⇒ void`

Write provided value as byte to the byte stream at current position.

`writeInt16B(value) ⇒ void`

Write provided value to the byte stream as big-endian 16-bit integer .

`writeInt32B(value) ⇒ void`

Write provided value to the byte stream as big-endian 32-bit integer.

`writeInt64B(value) ⇒ void`

Write provided value to the byte stream as big-endian 64-bit integer.

`writeFloatB(value) ⇒ void`

Write provided value to the byte stream as big-endian 64-bit float.

`writeInt16L(value) ⇒ void`

Write provided value to the byte stream as little-endian 16-bit integer.

`writeInt32L(value) ⇒ void`

Write provided value to the byte stream as little-endian 32-bit integer.

`writeInt64L(value) ⇒ void`

Write provided value to the byte stream as little-endian 64-bit integer.

`writeFloatL(value) ⇒ void`

Write provided value to the byte stream as little-endian 64-bit float.

`writeCString(string) ⇒ void`

Write provided string to the byte stream as null-terminated string (C-string).

`writePString(string) ⇒ void`

Write provided string to the byte stream as Pascal string (first byte is string length).

`writeString(string) ⇒ void`

Write provided string to the byte stream.

Constructors

`ByteStream()`

Creates new byte stream.

Return

ByteStream object.

Example

```
bytestream = new ByteStream();
bytestream->writeString("xyz");
bytestream->seek(0);
while (!bytestream->eos) {
  b = bytestream->readByte();
  print(d2x(b,2) .. " "); // prints "78 79 7A"
}
```

Chassis

Object represent chassis, extends DataCollectionTarget

Instance attributes

`controller ⇒ Node`

Chassis controller node

`controllerId ⇒ Integer`

Chassis controller's node id

`flags ⇒ Integer`

Bit flags

`rack ⇒ NetObj`

Will return Rack object as NetObj if chassis is added in rack

`rackId ⇒ Integer`

Will return Rack id if chassis is added in rack

`rackHeight ⇒ Integer`

Object height in rack

`rackPosition ⇒ Integer`

Object position in rack

Constants

Access point state

Description	Value
Bind under controller	0x00000001

Class

Class represent classes and methodes in NXSL script

Instance attributes

`attributes ⇒ Array`

String aray with all attribute names

`hierarchy ⇒ Array`

String aray with class inheritance hirarchy

`methods ⇒ Array`

String aray with all methode names

`name ⇒ Integer`

Name of the class

ClientSession

Object represents client session

Instance attributes

`clientInfo ⇒ String`

Client app info string

`clientType ⇒ Integer`

Client system type - desktop(0), web(1), mobile(2), tablet(3), application(4)

`flags ⇒ Integer`

Client flags

`id ⇒ Bigint`

ID of client session

`loginName ⇒ String`

Login name

`loginTime ⇒ Integer`

Login time

`name ⇒ String`

User name

`systemAccessRights ⇒ Bigint`

User access ritht bit flags

`user ⇒ User`

object of User

`userId ⇒ Integer`

User ID

`webServerAddress ⇒ String`

Address for web server if it is web type connection

`workstation ⇒ String`

Workstation information if available

Cluster

Object represent cluster, extends DataCollectionTarget

Instance attributes

`nodes ⇒ Array`

Will return all nodes of class Node, that are under this cluster

`zone ⇒ Zone`

Will return zone this cluster is under

`zoneUIN ⇒ Integer`

Will return zone UIN this cluster is under

Instance methods

`getResourceOwner(name) ⇒ Node`

Get node which currently owns named resource.

Parameters

name	String	Name of the resource.

Return

Node object instance which currently owns resource of `null` if failed.

`add(object) ⇒ void`

Add node to cluster

Parameters

object	Node	Object to be added to cluster

`remove(object) ⇒ void`

Remove node from cluster

Parameters

object	Node	Object to be removed from cluster

Component

```
Node components
```

Instance attributes

`class ⇒ String`

Type of the component:

- `unknown`
- `chassis`
- `backplane`
- `container`
- `power supply`

- `fan`
- `sensor`
- `module`
- `port`
- `stack`

`children ⇒ Array`

List of direct children (Array of Component object intances).

`description ⇒ String`

Component description

`firmware ⇒ String`

Component firmware version, if available.

`ifIndex ⇒ Number`

Interface index number

`model ⇒ String`

Component model number, if available.

`name ⇒ String`

Component name, if available.

`serial ⇒ String`

Component serial number, if available.

`vendor ⇒ String`

Component vendor, if available.

Container

Object represent container, extends NetObj.

Instance attributes

`autoBindScript ⇒ String`

Source of the script for automatic binding.

`isAutoBindEnabled ⇒ Boolean`

Indicate if automatic binding is enabled.

`isAutoUnbindEnabled ⇒ Boolean`

Indicate if automatic unbinding is enabled.

Instance methods

`setAutoBindMode(enableBind, enableUnbind) ⇒ void`

Set automatic bind mode for the container.

Parameters

enableBind	Boolean	Script should be used for automatic binding.
enableUnbind	Boolean	Script should be used for automatic unbinding.

`setAutoBindScript(script) ⇒ void`

Update automatic binding script source.

Parameters

script	String	Script source.

DataCollectionTarget

Abstract class that represents any object that can collect data. Extends NetObj.

Instance attributes

`templates ⇒ Array`

Returns array of templates (Template) applied on this object. Return value also affected by trusted nodes settings.

Example

```
// Log names and ids of all accessible templates for current node
templates = $node->templates;
foreach(t : templates)
{
    trace(1, "Template object: name='" . t->name . "' id=" . t->id);
}
```

Instance methods

`applyTemplate(template) ⇒ void`

This method is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

Apply template to node

Parameters

template	Template	Template object to apply

`enableConfigurationPolling(flag) ⇒ void`

Enable or disable configuration polling for a node

Parameters

flag	Boolean	If configuration polling should be enabled.

`enableDataCollection(flag) ⇒ void`

Enable or disable data collection polling for a node.

Parameters

flag	Boolean	If data collection polls should be enabled.

`enableStatusPolling(flag) ⇒ void`

Enable or disable status polling for a node.

Parameters

flag	Boolean	If status polls should be enabled.

`readInternalParameter(name) ⇒ String`

Reads object internal metric (metric with source "Internal").

Parameters

name	String	Metric name.

`readInternalTable(name) ⇒ String`

Reads object internal table metric (metric with source "Internal").

Parameters

name	String	Metric name.

`removeTemplate(template) ⇒ void`

This method is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

Remove template from node

Parameters

template	Template	Template object to remove

DCI

Represents Data Collection Item (DCI).

Instance attributes

`activeThresholdSeverity ⇒ Integer`

Severity of the active threshold. If there are no active thresholds, defaults to `0` (NORMAL).

`comments ⇒ String`

DCI Comments (since 2.0-M5)

`dataType ⇒ Integer`

Data type of the DCI.

`description ⇒ String`

Description

`errorCount ⇒ Integer`

Number of consecutive data collection errors

`flags ⇒ Integer`

DCI flags

`hasActiveThreshold ⇒ Boolean`

`TRUE` if DCI has active threshold

`id ⇒ Integer`

Unique DCI identifier

`instance ⇒ String`

DCI instance (only for single value DCIs): %{instance}

`instanceName ⇒ String`

DCI instance name (only for single value DCIs): %{instance-name}

`lastPollTime ⇒ Integer64`

Time of last DCI poll (either successful or not) as number of seconds since epoch (1 Jan 1970 00:00:00 UTC)

`lastCollectionTime ⇒ Integer64`

Time of last successful DCI poll as number of seconds since epoch (1 Jan 1970 00:00:00 UTC). Added in version 4.0.

`name ⇒ String`

Parameter's name

`origin ⇒ Integer`

Data origin of the DCI.

`pollingInterval ⇒ Integer`

Current polling interval

`relatedObject ⇒ NetObj`

Related object or null if there is no object

`status ⇒ Integer`

Data status of the DCI.

`systemTag ⇒ String`

System tag. Always empty for user-defined DCIs.

`template ⇒ Template`

Template object if DCI came from Template or `NULL`.

`templateId ⇒ Integer`

Template id if DCI came from Template or 0.

`templateItemId ⇒ Integer`

DCI item id in template if DCI came from Template or 0.

`type ⇒ Integer`

DCI type:

- 1 = single value
- 2 = table

Instance methods

`forcePoll() ⇒ void`

Start DCI force poll.

DiscoveredInterface

Represent discovered interface objects in discovered node class

Instance attributes

`alias ⇒ String`

Interface alias (usually value of SNMP ifAlias).

`chassis ⇒ Integer`

Chassis id

`description ⇒ String`

Interface description

`index ⇒ Integer`

Interface index.

`ipAddressList ⇒ Array`

Array with InetAddress objects, that represent all adresses that has this interface has

`isPhysicalPort ⇒ Boolean`

`TRUE` if this interface object represents physical port

`macAddr ⇒ String`

String representation of MAC address separated by ":".

`module ⇒ Integer`

Module

`mtu ⇒ Integer`

Interface MTU (0 if unknown).

`name ⇒ String`

Interface name

`pic ⇒ Integer`

Phisical location.

`port ⇒ Integer`

Port number.

`speed ⇒ Integer64`

Speed of the interface.

`type ⇒ Integer`

Interface type

DiscoveredNode

Represents NetXMS node found while **performing** network discovery. Is available as a $node in script that is set as DiscoveryFilter.

Instance attributes

`agentVersion ⇒ String`

NetXMS agent version as `string`.

`dnsName ⇒ String`

Node's DNS name

`interfaces ⇒ Array`

Array with node's interfaces

`ipAddr ⇒ String`

Node's ip address

`isAgent ⇒ Boolean`

TRUE if NetXMS agent detected on node

`isBridge ⇒ Boolean`

TRUE if node is a bridge

`isCDP ⇒ Boolean`

TRUE if node supports CDP (Cisco Discovery Protocol)

`isLLDP ⇒ Boolean`

TRUE if node supports LLDP (Link Layer Discovery Protocol)

`isPrinter ⇒ Boolean`

TRUE if node is a printer

`isRouter ⇒ Boolean`

TRUE if node is a router (has IP forwarding enabled)

`isSNMP ⇒ Boolean`

TRUE if SNMP agent detected on node

`isSONMP ⇒ Boolean`

TRUE if node supports SONMP/NDP (Synoptics/Nortel Discovery Protocol)

`isSSH ⇒ Boolean`

TRUE if SSH connection is available for this node

`netMask ⇒ Integer`

Network mask

`platformName ⇒ String`

Platform name reported by NetXMS agent

`snmpOID ⇒ String`

SNMP object identifier (result of `.1.3.6.1.2.1.1.2.0` request)

`snmpVersion ⇒ Integer`

Configured SNMP version:

- 0: SNMP version 1
- 1: SNMP version 2c
- 2: SNMP version 3

`subnet ⇒ String`

Subnet

`zone ⇒ Zone`

Zone object (`null` if zoning is disabled)

`zoneUIN ⇒ Integer`

This node zone UIN

Event

Represents NetXMS event object.

Instance attributes

`code ⇒ Number`

Event code

`customMessage ⇒ String`

Custom message set in event processing policy by calling `setMessage`. Deprecated and will be removed in a future version.

`dci ⇒ DCI`

DCI object of class DCI that is source for this event or NULL if generated not by threshold

`dciId ⇒ Number`

DCI id that is source for this event or 0 if generated not by threshold

`id ⇒ Number`

Unique event identifier.

`lastAlarmKey ⇒ String`

Key for last generated alarm for this event or null

`message ⇒ String`

Event message. Get/set attribute.

`name ⇒ String`

Event name.

`origin ⇒ Number`

Origin of the event

- 0 - SYSTEM
- 1 - AGENT
- 2 - CLIENT
- 3 - SYSLOG
- 4 - SNMP
- 5 - NXSL
- 6 - REMOTE_SERVER

`originTimestamp ⇒ Number`

The time when the event was generated in the origin.

`parameters ⇒ Array`

List of event parameters. Starting index is 1.

`parameterNames ⇒ Array`

List of named event parameters (e.g. "dciId"), which can be accessed by `object->parameterName`.

`rootId ⇒ Number64`

Id of event that is root cause of this event or 0 if there is no such event.

`severity ⇒ Number`

Event severity code. Get/set attribute.

`source ⇒ NetObj`

Source object (inherited from NetObj, exact type can be checked with classof() function) for the event.

`sourceId ⇒ Number`

ID of the source object for the event.

`tags ⇒ Array`

Event tags as an array of strings.

`tagList ⇒ String`

Event tags as a coma separated list.

`timestamp ⇒ Integer`

Unix timestamp of the event.

`$1…$n`

Shortcut for `parameters[n]` (e.g. "$event→parameters[3]" can be replaced with "$event→$3").

`$…`

Named event parameters can be accessed directly by the name (e.g `$event->dciId`). List of available named parameters can be accessed with `parameterNames` attribute. Get/set attribute.

Instance methods

`addParameter(name, value) ⇒ void`

Set event parameter

Parameters

name	String	Parameter name. Optional parameter.
value	String	Parameter value.

`addTag(tag) ⇒ void`

Set event tag, which can be later accessed via `tags` attribute.

Parameters

tag	String tag

`correlateTo(eventId) ⇒ void`

Sets root cause id for the event

Parameters

eventId	Root cause event id

`expandString(string) ⇒ String`

Expand string by replacing macros with their values.

Parameters

string	String to expand

Return

Formated string

Example

```
>>> $event->expandString("%N")
SYS_THRESHOLD_REACHED
```

`hasTag(tag) ⇒ Boolean`

Return if event has specific tag.

Parameters

tag	String tag

`removeTag(tag) ⇒ void`

Remove tag from event tag list

Parameters

tag	String tag

`setMessage(message) ⇒ void`

Set event message to `message`.

Parameters

message	Message string

`setNamedParameter(name, value) ⇒ void`

Set named parameter or change value of existing named parameter

Parameters

name	String name of parameter
value	String value of parameter

`setParameter(index, value) ⇒ void`

Set value of parameter or change value of existing parameter. If index corresponds to existing parameter, the value will be set. If index is higher then existing number of parameters, new parameter(s) will be added (if needed, parameters with empty values will be added to reach the number of parameters indicated by index).

Parameters

index	Integer index of parameter
value	String value of parameter

`setSeverity(severityCode) ⇒ void`

Change event severity to `severityCode`.

Parameters

severityCode	Numeric severity code

`toJson() ⇒ String`

Serialize object to JSON.

Return

String representation of the object in JSON format.

FILE

Class that represents file on file system. Available only if 'NXSL.EnableFileIOFunctions' configuration parameter is turned on. File read and write position in file depends on mode file was opened in.

Instance attributes

`eof ⇒ Boolean`

If it's already end of file.

`name ⇒ String`

File name

Instance methods

`close() ⇒ void`

Close file

`read(numberOfSymbols) ⇒ String`

Read provided number of bytes from file.

Parameters

numberOfSymbols	Integer	Number of symbol to read from file

Return

String with symbols read from file or null if file closed or end of file reached.

`readLine() ⇒ String`

Read line from file.

Return

String with read line from file or null if file closed or end of file reached.

`write(text) ⇒ void`

Will write provided text to the file.

Parameters

text	String	Text to write in to the file

`writeLine(text) ⇒ void`

Will write provided text to the file and will add new line at the end.

Parameters

text	String	Text to write in to the file

GeoLocation

Represents geographical location (defined by latitude and longitude).

Instance attributes

`isManual ⇒ Boolean`

`TRUE` if location is entered manually

`isValid ⇒ Boolean`

`TRUE` if location is valid

`latitude ⇒ Number`

Latitude as floating point number

`latitudeText ⇒ String`

Latitude as text

`longitude ⇒ Number`

Longitude as floating point number

`longitudeText ⇒ String`

Longitude as text

`type ⇒ Number`

Data source type:

- 0 – Unset
- 1 – Manual
- 2 - GPS
- 3 - Network

Constructors

`GeoLocation(latitude, longitude, type=1)`

Create instance of the class based on floating-point `latitude` and `longitude`. Optional argument `type` can be used to override default value 1 ("Manual").

Constants

Location Types

Value	Description
0	Unset (represents unknown location)
1	Manual (set by system administrator)
2	Automatic (obtained from GPS)
3	Network (obtained from network, for example using WiFi AP database)

Examples

Print node location

```
>>> nodeLoc = $node->geolocation
>>> println(nodeLoc->latitudeText)
N 48° 00' 0.000"
>>> println(nodeLoc->longitudeText)
E 22° 00' 0.000"
```

Set node location

```
>>> nodeLoc = GeoLocation(22.11, 48.12, 1)
>>> $node->setGeoLocation(nodeLoc)
>>> println($node->geolocation->latitudeText)
N 48° 12' 0.000"
>>> println($node->geolocation->longitudeText)
E 22° 11' 0.000"
```

Clear location

```
>>> $node->clearGeoLocation()
>>> println($node->geolocation)
null
```

HardwareComponent

Object that contains all information about node hardware component.

Instance Attributes

`changeCode ⇒ Integer`

Code of the changes that happened to the component during the last configuration poll.

Possible values are listed here: Change Code

`categoryName ⇒ String`

Component category name

`category ⇒ Integer`

Code of the component category.

Possible values are:

Constant	Value
HardwareComponentCategory::OTHER	0
HardwareComponentCategory::BASEBOARD	1
HardwareComponentCategory::PROCESSOR	2
HardwareComponentCategory::MEMORY	3
HardwareComponentCategory::STORAGE	4
HardwareComponentCategory::BATTERY	5
HardwareComponentCategory::NETWORK_ADAPTER	6

`index ⇒ Integer`

Index of the component.

`capacity ⇒ Integer`

Component capacity depending on the type.

`type ⇒ String`

Component type depending on the category.

`vendor ⇒ String`

Component vendor.

`model ⇒ String`

Component model.

`location ⇒ String`

Component installation location in the system.

`partNumber ⇒ String`

Component part number.

`serialNumber ⇒ String`

Component serial number.

`description ⇒ String`

Component description.

InetAddress

Object that contains all information about network address

Instance attributes

`address ⇒ String`

IP address

`family ⇒ String`

Internet address, one of:

- inet
- inet6
- unspec

`isAnyLocal ⇒ Boolean`

`TRUE` if address is a wildcard address

`isBroadcast ⇒ Boolean`

TRUE if address is a broadcast address

`isLinkLocal ⇒ Boolean`

TRUE if address is a link local address

`isLoopback ⇒ Boolean`

TRUE if address is a loopback address

`isMulticast ⇒ Boolean`

TRUE if address is a multicast address

`isValid ⇒ Boolean`

TRUE if address valid

`isValidUnicast ⇒ Boolean`

TRUE if address valid unicast

`mask ⇒ Integer`

Address bit mask

Constructors

`InetAddress()`

Constructor for internet address

Return

InetAddress object

`InetAddress(address)`

Constructor for internet address

Parameters

address	String	IP address as a string

Return

InetAddress object

Interface

Represent interface object. Inherit all attributes and methods of the NetObj class.

Instance attributes

`adminState ⇒ Integer`

Administrative state of the interface.

`bridgePortNumber ⇒ Integer`

Bridge port number for this interface.

`chassis ⇒ Chassis`

Parent Chassis

`description ⇒ String`

Interface description

`dot1xBackendAuthState ⇒ Integer`

802.1x back-end authentication state

`dot1xPaeAuthState ⇒ Integer`

802.1x PAE authentication state

`expectedState ⇒ Integer`

Expected state of the interface.

`flags ⇒ Integer`

Interface flags (bit mask, `uint32`).

`icmpAverageRTT ⇒ Integer`

ICMP average response time for current interface. Will return null if no information.

`icmpLastRTT ⇒ Integer`

ICMP last response time for current interface. Will return null if no information.

`icmpMaxRTT ⇒ Integer`

ICMP maximal response time for current interface. Will return null if no information.

`icmpMinRTT ⇒ Integer`

ICMP minimal response time for current interface. Will return null if no information.

`icmpPacketLoss ⇒ Integer`

ICMP packet loss for current interface. Will return null if no information.

`ifAlias ⇒ String`

Interface alias

`ifIndex ⇒ Integer`

Interface index.

`ifType ⇒ Integer`

Interface type

`ipAddressList ⇒ Array`

Array with InetAddress objects, that represent all addresses that has this interface has

`isExcludedFromTopology ⇒ Boolean`

TRUE if this interface excluded from network topology

`isIncludedInIcmpPoll ⇒ Boolean`

TRUE if this interface is included in ICMP statistics

`isLoopback ⇒ Boolean`

TRUE if this interface is a loopback

`isManuallyCreated ⇒ Boolean`

TRUE if this interface object was created manually by NetXMS administrator

`isOSPF ⇒ Boolean`

TRUE if this interface is OSPF

`isPhysicalPort ⇒ Boolean`

TRUE if this interface object represents physical port

`macAddr ⇒ String`

String representation of MAC address separated by ":".

`module ⇒ Integer`

Module

`mtu ⇒ Integer`

Interface MTU (0 if unknown).

`node ⇒ Node`

Parent node object

`operState ⇒ Integer`

Operational state.

`ospfAreaId ⇒ String`

OSPF area ID

`ospfState ⇒ Integer`

OSPF state

`ospfStateText ⇒ String`

OSPF state as a text

`ospfType ⇒ Integer`

OSPF type

`ospfTypeText ⇒ String`

OSPF type as s text

`peerInterface ⇒ Interface`

Peer interface object if known, otherwise `null`.

`peerNode ⇒ Node`

Peer node object if known, otherwise `null`.

`pic ⇒ Integer`

Phisical location.

`port ⇒ Integer`

Port number.

`speed ⇒ Integer64`

Speed of the interface.

`vlans ⇒ Array`

Array with this inteface vlan objects

`zone ⇒ Zone`

Zone object (null if zoning is disabled).

`zoneUIN ⇒ Integer`

Zone UIN of this interface.

Instance methods

`enableAgentStatusPolling(enabled) ⇒ void`

Enable\disable agent status polling for this interface

Parameters

enabled	Boolean	`TRUE` if interface should be status polled by NetXMS agent

`enableICMPStatusPolling(enabled) ⇒ void`

Enable\disable ICMP status polling for this interface

Parameters

enabled	Boolean	`TRUE` if interface should be status polled by ICMP

`enableSNMPStatusPolling(enabled) ⇒ void`

Enable\disable SNMP status polling for this interface

Parameters

enabled	Boolean	`TRUE` if interface should be status polled by SNMP

`setExcludeFromTopology(excluded) ⇒ void`

Change `isExcludedFromTopology` flag.

Parameters

excluded	Boolean	`TRUE` if interface should be excluded.

`setExpectedState(newState) ⇒ void`

Set expected state to `newState`.

Parameters

newState	Number	New state as defined by Interface expected states.

`setIncludeInIcmpPoll(enabled) ⇒ void`

Enabele/Disable ICMP statistics collection for current interface.

Parameters

enabled	Boolean	If this interface should be included in ICMP statistics.

Constants

Interface states

Code	Description
0	Unknown
1	Up
2	Down
3	Testing

Interface expected states

Code	Description
0	Up
1	Down
2	Ignore

Interface types

Code	Type
1	IFTYPE_OTHER
2	IFTYPE_REGULAR1822
3	IFTYPE_HDH1822
4	IFTYPE_DDN_X25
5	IFTYPE_RFC877_X25
6	IFTYPE_ETHERNET_CSMACD
7	IFTYPE_ISO88023_CSMACD

Code	Type
8	IFTYPE_ISO88024_TOKENBUS
9	IFTYPE_ISO88025_TOKENRING
10	IFTYPE_ISO88026_MAN
11	IFTYPE_STARLAN
12	IFTYPE_PROTEON_10MBIT
13	IFTYPE_PROTEON_80MBIT
14	IFTYPE_HYPERCHANNEL
15	IFTYPE_FDDI
16	IFTYPE_LAPB
17	IFTYPE_SDLC
18	IFTYPE_DS1
19	IFTYPE_E1
20	IFTYPE_BASIC_ISDN
21	IFTYPE_PRIMARY_ISDN
22	IFTYPE_PROP_PTP_SERIAL
23	IFTYPE_PPP
24	IFTYPE_SOFTWARE_LOOPBACK
25	IFTYPE_EON
26	IFTYPE_ETHERNET_3MBIT
27	IFTYPE_NSIP
28	IFTYPE_SLIP
29	IFTYPE_ULTRA
30	IFTYPE_DS3
31	IFTYPE_SMDS
32	IFTYPE_FRAME_RELAY
33	IFTYPE_RS232
34	IFTYPE_PARA
35	IFTYPE_ARCNET
36	IFTYPE_ARCNET_PLUS
37	IFTYPE_ATM
38	IFTYPE_MIOX25
39	IFTYPE_SONET
40	IFTYPE_X25PLE

Code	Type
41	IFTYPE_ISO88022LLC
42	IFTYPE_LOCALTALK
43	IFTYPE_SMDS_DXI
44	IFTYPE_FRAME_RELAY_SERVICE
45	IFTYPE_V35
46	IFTYPE_HSSI
47	IFTYPE_HIPPI
48	IFTYPE_MODEM
49	IFTYPE_AAL5
50	IFTYPE_SONET_PATH
51	IFTYPE_SONET_VT
52	IFTYPE_SMDS_ICIP
53	IFTYPE_PROP_VIRTUAL
54	IFTYPE_PROP_MULTIPLEXOR
55	IFTYPE_IEEE80212
56	IFTYPE_FIBRECHANNEL
57	IFTYPE_HIPPIINTERFACE
58	IFTYPE_FRAME_RELAY_INTERCONNECT
59	IFTYPE_AFLANE8023
60	IFTYPE_AFLANE8025
61	IFTYPE_CCTEMUL
62	IFTYPE_FAST_ETHERNET
63	IFTYPE_ISDN
64	IFTYPE_V11
65	IFTYPE_V36
66	IFTYPE_G703_AT64K
67	IFTYPE_G703_AT2MB
68	IFTYPE_QLLC
69	IFTYPE_FASTETHERFX
70	IFTYPE_CHANNEL
71	IFTYPE_IEEE80211
72	IFTYPE_IBM370_PARCHAN
73	IFTYPE_ESCON

Code	Type
74	IFTYPE_DLSW
75	IFTYPE_ISDNS
76	IFTYPE_ISDNU
77	IFTYPE_LAPD
78	IFTYPE_IPSWITCH
79	IFTYPE_RSRB
80	IFTYPE_ATMLOGICAL
81	IFTYPE_DS0
82	IFTYPE_DS0_BUNDLE
83	IFTYPE_BSC
84	IFTYPE_ASYNC
85	IFTYPE_CNR
86	IFTYPE_ISO88025DTR
87	IFTYPE_EPLRS
88	IFTYPE_ARAP
89	IFTYPE_PROPCNLS
90	IFTYPE_HOSTPAD
91	IFTYPE_TERMPAD
92	IFTYPE_FRAME_RELAY_MPI
93	IFTYPE_X213
94	IFTYPE_ADSL
95	IFTYPE_RADSL
96	IFTYPE_SDSL
97	IFTYPE_VDSL
98	IFTYPE_ISO88025CRFPINT
99	IFTYPE_MYRINET
100	IFTYPE_VOICEEM
101	IFTYPE_VOICEFXO
102	IFTYPE_VOICEFXS
103	IFTYPE_VOICEENCAP
104	IFTYPE_VOICEOVERIP
105	IFTYPE_ATMDXI
106	IFTYPE_ATMFUNI

Code	Type
107	IFTYPE_ATMIMA
108	IFTYPE_PPPMULTILINKBUNDLE
109	IFTYPE_IPOVERCDLC
110	IFTYPE_IPOVERCLAW
111	IFTYPE_STACKTOSTACK
112	IFTYPE_VIRTUAL_IP_ADDRESS
113	IFTYPE_MPC
114	IFTYPE_IPOVERATM
115	IFTYPE_ISO88025FIBER
116	IFTYPE_TDLC
117	IFTYPE_GIGABIT_ETHERNET
118	IFTYPE_HDLC
119	IFTYPE_LAPF
120	IFTYPE_V37
121	IFTYPE_X25MLP
122	IFTYPE_X25_HUNT_GROUP
123	IFTYPE_TRANSPHDLC
124	IFTYPE_INTERLEAVE
125	IFTYPE_FAST
126	IFTYPE_IP
127	IFTYPE_DOCSCABLE_MACLAYER
128	IFTYPE_DOCSCABLE_DOWNSTREAM
129	IFTYPE_DOCSCABLE_UPSTREAM
130	IFTYPE_A12MPPSWITCH
131	IFTYPE_TUNNEL
132	IFTYPE_COFFEE
133	IFTYPE_CES
134	IFTYPE_ATM_SUBINTERFACE
135	IFTYPE_L2VLAN
136	IFTYPE_L3IPVLAN
137	IFTYPE_L3IPXVLAN
138	IFTYPE_DIGITAL_POWERLINE
139	IFTYPE_MEDIAMAIL_OVER_IP

Code	Type
140	IFTYPE_DTM
141	IFTYPE_DCN
142	IFTYPE_IPFORWARD
143	IFTYPE_MSDSL
144	IFTYPE_IEEE1394
145	IFTYPE_GSN
146	IFTYPE_DVBRCC_MACLAYER
147	IFTYPE_DVBRCC_DOWNSTREAM
148	IFTYPE_DVBRCC_UPSTREAM
149	IFTYPE_ATM_VIRTUAL
150	IFTYPE_MPLS_TUNNEL
151	IFTYPE_SRP
152	IFTYPE_VOICE_OVER_ATM
153	IFTYPE_VOICE_OVER_FRAME_RELAY
154	IFTYPE_IDSL
155	IFTYPE_COMPOSITE_LINK
156	IFTYPE_SS7_SIGLINK
157	IFTYPE_PROPWIRELESSP2P
158	IFTYPE_FRFORWARD
159	IFTYPE_RFC1483
160	IFTYPE_USB
161	IFTYPE_IEEE8023ADLAG
162	IFTYPE_BGP_POLICY_ACCOUNTING
163	IFTYPE_FRF16MFR_BUNDLE
164	IFTYPE_H323_GATEKEEPER
165	IFTYPE_H323_PROXY
166	IFTYPE_MPLS
167	IFTYPE_MFSIGLINK
168	IFTYPE_HDSL2
169	IFTYPE_SHDSL
170	IFTYPE_DS1FDL
171	IFTYPE_POS
172	IFTYPE_DVBASI_IN

Code	Type
173	IFTYPE_DVBASI_OUT
174	IFTYPE_PLC
175	IFTYPE_NFAS
176	IFTYPE_TR008
177	IFTYPE_GR303RDT
178	IFTYPE_GR303IDT
179	IFTYPE_ISUP
180	IFTYPE_PROPDOCSWIRELESSMACLAYER
181	IFTYPE_PROPDOCSWIRELESSDOWNSTREAM
182	IFTYPE_PROPDOCSWIRELESSUPSTREAM
183	IFTYPE_HIPERLAN2
184	IFTYPE_PROPBWAP2MP
185	IFTYPE_SONET_OVERHEAD_CHANNEL
186	IFTYPE_DW_OVERHEAD_CHANNEL
187	IFTYPE_AAL2
188	IFTYPE_RADIOMAC
189	IFTYPE_ATMRADIO
190	IFTYPE_IMT
191	IFTYPE_MVL
192	IFTYPE_REACHDSL
193	IFTYPE_FRDLCIENDPT
194	IFTYPE_ATMVCIENDPT
195	IFTYPE_OPTICAL_CHANNEL
196	IFTYPE_OPTICAL_TRANSPORT
197	IFTYPE_PROPATM
198	IFTYPE_VOICE_OVER_CABLE
199	IFTYPE_INFINIBAND
200	IFTYPE_TELINK
201	IFTYPE_Q2931
202	IFTYPE_VIRTUALTG
203	IFTYPE_SIPTG
204	IFTYPE_SIPSIG
205	IFTYPE_DOCSCABLEUPSTREAMCHANNEL

Code	Type
206	IFTYPE_ECONET
207	IFTYPE_PON155
208	IFTYPE_PON622
209	IFTYPE_BRIDGE
210	IFTYPE_LINEGROUP
211	IFTYPE_VOICEEMFGD
212	IFTYPE_VOICEFGDEANA
213	IFTYPE_VOICEDID
214	IFTYPE_MPEG_TRANSPORT
215	IFTYPE_SIXTOFOUR
216	IFTYPE_GTP
217	IFTYPE_PDNETHERLOOP1
218	IFTYPE_PDNETHERLOOP2
219	IFTYPE_OPTICAL_CHANNEL_GROUP
220	IFTYPE_HOMEPNA
221	IFTYPE_GFP
222	IFTYPE_CISCO_ISL_VLAN
223	IFTYPE_ACTELIS_METALOOP
224	IFTYPE_FCIPLINK
225	IFTYPE_RPR
226	IFTYPE_QAM
227	IFTYPE_LMP
228	IFTYPE_CBLVECTASTAR
229	IFTYPE_DOCSCABLEMCMTSDOWNSTREAM
230	IFTYPE_ADSL2
231	IFTYPE_MACSECCONTROLLEDIF
232	IFTYPE_MACSECUNCONTROLLEDIF
233	IFTYPE_AVICIOPTICALETHER
234	IFTYPE_ATM_BOND
235	IFTYPE_VOICEFGDOS
236	IFTYPE_MOCA_VERSION1
237	IFTYPE_IEEE80216WMAN
238	IFTYPE_ADSL2PLUS

Code	Type
239	IFTYPE_DVBRCSMACLAYER
240	IFTYPE_DVBTDM
241	IFTYPE_DVBRCSTDMA
242	IFTYPE_X86LAPS
243	IFTYPE_WWANPP
244	IFTYPE_WWANPP2
245	IFTYPE_VOICEEBS
246	IFTYPE_IFPWTYPE
247	IFTYPE_ILAN
248	IFTYPE_PIP
249	IFTYPE_ALUELP
250	IFTYPE_GPON
251	IFTYPE_VDSL2
252	IFTYPE_CAPWAP_DOT11_PROFILE
253	IFTYPE_CAPWAP_DOT11_BSS
254	IFTYPE_CAPWAP_WTP_VIRTUAL_RADIO
255	IFTYPE_BITS
256	IFTYPE_DOCSCABLEUPSTREAMRFPORT
257	IFTYPE_CABLEDOWNSTREAMRFPORT
258	IFTYPE_VMWARE_VIRTUAL_NIC
259	IFTYPE_IEEE802154
260	IFTYPE_OTNODU
261	IFTYPE_OTNOTU
262	IFTYPE_IFVFITYPE
263	IFTYPE_G9981
264	IFTYPE_G9982
265	IFTYPE_G9983
266	IFTYPE_ALUEPON
267	IFTYPE_ALUEPONONU
268	IFTYPE_ALUEPONPHYSICALUNI
269	IFTYPE_ALUEPONLOGICALLINK
270	IFTYPE_ALUGPONONU
271	IFTYPE_ALUGPONPHYSICALUNI

Code	Type
272	IFTYPE_VMWARE_NIC_TEAM

JsonArray

Represents JSON array object

Instance attributes

`size ⇒ Number`

Number of elements in array

`values ⇒ Array`

NXSL array with JSON array elements

Instance methods

`append(value) ⇒ void`

Appends value to JSON array.

Parameters

value	?	Add value to JSON array. Value type may vary.

`get(index) ⇒ ?`

Returns array value by the index. Value type depends on the type used in JSON.

Parameters

index	Integer	Index of element in array

Return

Attribute value

`insert(index, value) ⇒ void`

Sets value to the provided index in JSON array, moving existing element in this position

Parameters

index	Integer	Index of element in array
value	?	Add value to JSON array. Value type may vary.

`serialize() ⇒ String`

Returns string with serialized JSON

Return

String with serialized JSON

`set(index, value) ⇒ void`

Sets value to the provided index in JSON array, in place of existing element in this position

Parameters

index	Integer	Index of element in array
value	?	Add value to JSON array. Value type may vary.

Constructors

`JsonArray()`

Creates new JSON array.

Return

JsonArray object

JsonObject

Represents JSON object

Instance attributes

Attribute values can be accessed in the same way as instance attribute.

Instance methods

`get(key) ⇒ ?`

Returns attribute value by the key. Value type depends on the type used in JSON.

Parameters

key	String	Attribute key

Return

Attribute value

`keys() ⇒ Array`

Returns attribute array

Return

Attribute array

`serialize() ⇒ String`

Returns string with serialized JSON

Return

String with JSON

`set(key, value) ⇒ void`

Sets attribute referenced by key to the given value.

Parameters

key	String	Attribute key
value	String	Attribute new value

Constructors

`JsonObject()`

Creates new JSON object.

Return

JsonObject object

LDAPObject

Represents object recieved form LDAP on synchronization

Instance attributes

`description ⇒ String`

Description

`email ⇒ String`

E-mail

`fullName ⇒ String`

Full name

`id ⇒ String`

Id

`isGroup ⇒ Boolean`

If objec is a group

`isUser ⇒ Boolean`

If object is an user

`loginName ⇒ String`

Login name

`phoneNumber ⇒ String`

Phone number

`type ⇒ String`

String that repsesents type. One of: group, user, other

MaintenanceJournalRecord

One maintenance journal record entry

Instance attributes

`authorId ⇒ Integer`

Autor user ID

`creationTime ⇒ Integer`

UNIX timestamp of creation

`description ⇒ String`

Description

`editorId ⇒ Integer`

Editor user ID

`modificationTime ⇒ Integer`

UNIX timestamp of last modification

`objectId ⇒ Integer`

ID of an object that was under maintenance

MobileDevice

Class that represents mobile device object, extends DataCollectionTarget

Instance attributes

`altitude ⇒ Integer`

Altitude

`batteryLevel ⇒ Integer`

Battery percentage

`commProtocol ⇒ String`

Communication Protocol

`deviceId ⇒ String`

Device id

`direction ⇒ String`

Direction

`lastReportTime ⇒ Integer`

UNIX timestamp when device last reported data

`model ⇒ String`

Device model

`osName ⇒ String`

OS name installed on device

`osVersion ⇒ String`

OS version

`serialNumber ⇒ String`

Serial number

`speed ⇒ String`

Speed

`userId ⇒ String`

User id

`vendor ⇒ String`

Vendor

NetObj

Base class for all NetXMS objects.

Instance attributes

Object custom attributes can be accessed in the same way as instance attribute. If name of the custom attribute overlaps with the instance attribute, method NetObj::getCustomAttribute() should be used instead.

`alarms ⇒ array`

List of active Alarms for this object.

`alias ⇒ String`

Object alias. For interfaces it is automatically collected on configuration poll (e.g. from SNMP ifAlias).

`asset ⇒ Asset`

Reference to asset class.

`assetId ⇒ Integer`

Id of linked asset object.

`assetProperties ⇒ Asset Properties`

Reference to asset properties class.

`backupZoneProxy ⇒ Node`

Currently selected backup zone proxy (`null` if zoning is disabled or backup proxy is not assigned)

`backupZoneProxyId ⇒ Integer`

ID of currently selected backup zone proxy (`0` if zoning is disabled or backup proxy is not assigned)

`category ⇒ String`

Object's category name

`categoryId ⇒ Integer`

Object's category ID

`children ⇒ array`

List of child objects (inherited from NetObj). Use classof() to differentiate.

`city ⇒ String`

Postal address - city.

`comments ⇒ String`

Object comments.

`country ⇒ String`

Postal address - country.

`creationTime ⇒ Integer64`

Object creation time as UINX timestamp

`customAttributes ⇒ HashMap`

Hash map of object custom attributes.

`district ⇒ String`

Postal address - district.

`geolocation ⇒ GeoLocation`

Object geographical location.

`guid ⇒ String`

Object GUID as `string`.

`id ⇒ Integer`

Unique object identifier.

`ipAddr ⇒ String`

Primary IP address.

`isInMaintenanceMode ⇒ Boolean`

Maintenance mode indicator (`true` if object currently is in maintenace mode).

`maintenanceInitiator ⇒ Integer`

Maintenance initiator user id

`mapImage ⇒ String`

GUID of object image used for representation on the maps.

`name ⇒ String`

Object name.

`nameOnMap ⇒ String`

Object name displayed on map.

`parents ⇒ array`

List of direct parents for this object (inherited from NetObj, most likely either Container or Cluster).

`postcode ⇒ String`

Postal address - postal code.

`primaryZoneProxy ⇒ Node`

currently selected primary zone proxy (`null` if zoning is disabled or primary proxy is not assigned)

`primaryZoneProxyId ⇒ Integer`

ID of currently selected primary zone proxy (`0` if zoning is disabled or primary proxy is not assigned)

`region ⇒ String`

Postal address - region.

`responsibleUsers ⇒ Array`

Array with user objects that are added as responsible users for this object. Objects are User or UserGroup

`state ⇒ Integer`

Current object state. One of: Node state, Cluster state, Sensor state

`status ⇒ Integer`

Current object status.

`streetAddress ⇒ String`

Postal address - street.

`type ⇒ Integer`

Object type.

Instance methods

`bind(childObject) ⇒ void`

This method is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

Bind `childObject` to the current object as a child.

Parameters

object	NetObj	Object to bind as a child to the current object.

`bindTo(parentObject) ⇒ void`

This method is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

Bind current object to `parentObject` as a child.

Parameters

object	NetObj	Object to bind as a parent to the current object.

`clearGeoLocation() ⇒ void`

Clears GeoLocation data from the node

`delete() ⇒ void`

Deletes current object.

`deleteCustomAttribute(name) ⇒ void`

Delete custom attribute.

Parameters

name	String	Name of the attribute to delete.

`getResponsibleUsers(level) ⇒ array`

Returns responsible users that have escalation level equal to specified level.

Parameters

level	Integer

Return

Array with user objects that are added as responsible users for this object. Objects are User or UserGroup

`isDirectChild(object) ⇒ Boolean`

If provided object is direct child of this object

Parameters

object	NetObj

Return

TRUE if provided object is direct child of this object

`isDirectParent(object) ⇒ Boolean`

If provided object is direct parent of this object

Parameters

object	NetObj

Return

TRUE if provided object is direct parent of this object

`enterMaintenance() ⇒ void`

Enable maintenance mode for the object.

`expandString(string) ⇒ String`

Expand string by replacing macros with their values.

Parameters

string	String to expand

Return

Formated string

Example

```
>>> $node->expandString("%n")
My node name
```

`getCustomAttribute(name) ⇒ String`

Returns value of the custom attribute with the provided name.

Parameters

name	String	Name of the attribute to get value from.

Alternatively, attributes can be accessed as instance attribues (with `→`, attribute should exist) or by using `attribute@object` notion (which will return `null` instead of runtime error if attribue is missing).

`isChild(object) ⇒ Boolean`

If provided object is child of this object

Parameters

object	NetObj

Return

TRUE if provided object is child of this object

`isParent(object) ⇒ Boolean`

If provided object is parent of this object

Parameters

object	NetObj

Return

TRUE if provided object is parent of this object

`leaveMaintenance() ⇒ void`

Disable maintenance mode for the object.

`manage() ⇒ void`

Sets object to managed state. Has no affect if object already managed.

`readMaintenanceJournal(startTime, endTime) ⇒ Array`

Read maintenance entries

Parameters

startTime	Integer
Period start	endTime
Integer	Period end

Return

Array with MaintenanceJournalRecord objects

`rename(name) ⇒ void`

Rename object.

Parameters

name	String	New object name

`setAlias(name) ⇒ void`

Set object alias name

Parameters

name	String	New alias name

`setCategory(idOrName) ⇒ void`

Set object category id or name (used to get object display image)

Parameters

idOrName	String	ID or name of category

`setComments(comment) ⇒ void`

Set object comments

Parameters

comment	String	Comment to be set

`setCustomAttribute(key, value, inherit=false) ⇒ void`

Update or create custom attribute with the given key and value.

Parameters

key	String	Attribute key
value	String	Attribute value
inherit	Boolean	Optional parameter. If not set - inheritance will not be changed. `true` to inherit, `false` not to inherit.

`setGeoLocation(newLocation) ⇒ void`

Sets node geographical location.

Parameters

newLocation	GeoLocation

`setMapImage(image) ⇒ void`

Sets object image, that will be used to display object on network map

Parameters

image	String	GUID or name of image from image library

`setNameOnMap(name) ⇒ void`

Sets object's name, that will be used to display object on network map

Parameters

name	String	New object's name on map

`setStatusCalculation(type, ⋯) ⇒ void`

Sets status calculation method.

Parameters

type	Integer	Status calculation type. One of Status callculation types
...	Integer(s)	If single threshold or multiple thresholds type is selected, then threshold or thresholds in percentage should be provided as next parameters.

`setStatusPropagation(type, ⋯) ⇒ void`

Sets status propagation method.

Parameters

type	Integer	Status propagation type. One of Status propagation types
...	Integer(s)	For fixed value type - value (Object status codes) should be provided. For relative - offset should be provided. For severity - severity mapping should be provided (4 numbers Object status codes).

`unbind(object) ⇒ void`

This method is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

Unbind provided object from the current object.

Parameters

object	NetObj	Object to unbind from the current object.

`unbindFrom(object) ⇒ void`

This method is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

Unbind current object from the provided object.

Parameters

object	NetObj	Object to unbind from the current object.

`unmanage() ⇒ void`

Set object into unmanaged state. Has no effect if object is already in unmanaged state.

`writeMaintenanceJournal(description) ⇒ void`

Add entry to maintenance journal

Parameters

description	String	Message to be added

Constants

Object Types

Code	Description
0	Generic
1	Subnet

Code	Description
2	Node
3	Interface
4	Network
5	Container
6	Zone
7	Service Root
8	Template
9	Template Group
10	Template Root
11	Network Service
12	VPN Connector
13	Condition
14	Cluster
15	Business Service Prototype
16	Asset
17	Asset Group
18	Asset Root
19	Network Map Root
20	Network Map Group
21	Network Map
22	Dashboard Root
23	Dashboard
27	Business Service Root
28	Business Service
29	Collector
31	Mobile Device
32	Rack
33	Access Point
34	Wireless Domain
35	Chassis
36	Dashboard Group
37	Sensor

Status callculation types

Code	Description
0	Default
1	Most critical
2	Single threshold
3	Multiple thresholds

Status propagation types

Code	Description
0	Default
1	Unchanged
2	Fixed
3	Relative
4	Translated

Status propagation types

Code	Description
0	Default
1	Unchanged
2	Fixed
3	Relative
4	Translated

Node

Represents NetXMS node object. Extends DataCollectionTarget.

Instance attributes

`agentCertificateMappingData ⇒ String`

Agent certificate mapping data (set in object properties).

`agentCertificateMappingMethod ⇒ String`

Agent certificate mapping method (set in object properties).

`agentCertificateSubject ⇒ String`

Subject of certificate issued for agent tunnel on this node.

`agentId ⇒ String`

NetXMS agent unique ID (`string` representation of GUID). Will return all zeroes GUID if agent is not detected on node or does not have unique ID.

`agentProxy ⇒ Node`

Object that is set as agent proxy in object properties or NULL.

`agentVersion ⇒ String`

NetXMS agent version as `string`.

`bootTime ⇒ Integer64`

Number of seconds since node start or `0` if unknown.

`bridgeBaseAddress ⇒ String`

Base address of the switch formatted as 12 character `string` without separators. Value is only valid for bridges and switches. Special value `000000000000` indicates that address is unknown.

`capabilities ⇒ Integer`

Detected node capabilities ("Have Agent", "Support SNMP", etc.) Bitwise AND of Node capability flags constants.

`cipDeviceType ⇒ Integer`

EtherNet/IP device type

`cipDeviceTypeAsText ⇒ String`

EtherNet/IP device type as text

`cipExtendedStatus ⇒ Integer`

EtherNet/IP device extended status

`cipExtendedStatusAsText ⇒ String`

EtherNet/IP device extended status as text

`cipStatus ⇒ Integer`

EtherNet/IP device status

`cipStatusAsText ⇒ String`

EtherNet/IP device status as text

`cipState ⇒ Integer`

EtherNet/IP device state

`cipStateAsText ⇒ String`

EtherNet/IP device state as text

`cipVendorCode ⇒ Integer`

EtherNet/IP vendor code in numeric form.

`components ⇒ Component`

Entity MIB components of class Component.

`dependentNodes ⇒ Array`

Will return array with NodeDependency class objects. This array contains all objects that have current node as a proxy or data collection sources.

`driver ⇒ String`

Named of selected device-specific SNMP driver.

`downSince ⇒ Integer64`

UNIX timestamp when node went down.

`effectiveAgentProxy ⇒ Node`

Object that is effective agent proxy or NULL.

`effectiveIcmpProxy ⇒ Node`

Object that is effective ICMP proxy or NULL.

`effectiveSnmpProxy ⇒ Node`

Object that is effective SNMP proxy or NULL.

`flags ⇒ Integer`

Bit mask of Node flags.

`hasAgentIfXCounters ⇒ Boolean`

TRUE if agent supports 64-bit interface counters.

`hasEntityMIB ⇒ Boolean`

TRUE if supports ENTITY-MIB.

`hasIfXTable ⇒ Boolean`

TRUE if supports ifXTable via SNMP (64-bit counters for interface statistics).

`hasUserAgent ⇒ Boolean`

TRUE if has user agent

`hasVLANs ⇒ Boolean`

TRUE if VLAN information available.

`hardwareId ⇒ String`

Nodes' unique hardware id

`hardwareComponents ⇒ Array`

Retutrns an array of awailiable hardware of class HardwareComponent.

`hasWinPDH ⇒ Boolean`

TRUE if node supports Windows PDH parameters.

`hypervisorInfo ⇒ String`

Additional information about hypervisor for this node.

`hypervisorType ⇒ String`

Hypervisor type as `string` (usually hypervisor vendor or product name, like VMWare or XEN).

`icmpAverageRTT ⇒ Integer`

ICMP average response time for primary address. Will return null if no information.

`icmpLastRTT ⇒ Integer`

ICMP last response time for primary address. Will return null if no information.

`icmpMaxRTT ⇒ Integer`

ICMP maximal response time for primary address. Will return null if no information.

`icmpMinRTT ⇒ Integer`

ICMP minimal response time for primary address. Will return null if no information.

`icmpPacketLoss ⇒ Integer`

ICMP packet loss for primary address. Will return null if no information.

`icmpProxy ⇒ Node`

Object that is set as ICMP proxy in object properties or NULL.

`interfaces ⇒ Array`

Array with Interface objects, that are under this node. First object placed at index 0.

`is802_1x ⇒ Boolean`

`TRUE` if node supports 802.1x. Equivalent of `isPAE`.

`isAgent ⇒ Boolean`

`TRUE` if NetXMS agent detected on node

`isBridge ⇒ Boolean`

`TRUE` if node is a bridge

`isCDP ⇒ Boolean`

`TRUE` if node supports CDP (Cisco Discovery Protocol)

`isEtherNetIP ⇒ Boolean`

`TRUE` if node supports EtherNet/IP (Industrial Protocol)

`isExternalGateway ⇒ Boolean`

`TRUE` if node is remotely managed node

`isInMaintenanceMode ⇒ Boolean`

`TRUE` if node is in maintenance mode

`isLLDP ⇒ Boolean`

`TRUE` if node supports LLDP (Link Layer Discovery Protocol)

`isLocalManagement ⇒ Boolean`

TRUE if node is a local management server (NetXMS server)

`isLocalMgmt ⇒ Boolean`

TRUE if node is a local management server (NetXMS server)

`isNDP ⇒ Boolean`

TRUE if node supports OSPF/NDP. Equivalent of `isOSPF`.

`isModbusTCP ⇒ Boolean`

TRUE if node supports Modbus TCP

`isOSPF ⇒ Boolean`

TRUE if node supports OSPF/NDP. Equivalent of `isNDP`.

`isPAE ⇒ Boolean`

TRUE if node supports 802.1x. Equivalent of `is802_1x`.

`isPrinter ⇒ Boolean`

TRUE if node is a printer

`isProfiNet ⇒ Boolean`

TRUE if node supports PROFINET (Process Field Network)

`isRouter ⇒ Boolean`

TRUE if node is a router (has IP forwarding enabled)

`isSMCLP ⇒ Boolean`

TRUE if node supports SMCLP (Server Management Command Line Protocol)

`isSNMP ⇒ Boolean`

TRUE if SNMP agent detected on node

`isSSH ⇒ Boolean`

TRUE if SSH is connection available

`isSONMP ⇒ Boolean`

TRUE if node supports SONMP/NDP (Synoptics/Nortel Discovery Protocol)

`isSTP ⇒ Boolean`

TRUE if node supports STP (Spanning Tree Protocol)

`isUserAgentInstalled ⇒ Boolean`

TRUE if user agent is installed.

`isVirtual ⇒ Boolean`

TRUE if node is virtual

`isVRRP ⇒ Boolean`

TURE if VRRP supported.

`lastAgentCommTime ⇒ Integer`

Unix timestamp of last time when communication with agent was

`nodeSubType ⇒ String`

Node sub type

`nodeType ⇒ Integer`

Node type Node types

`ospfAreas ⇒ OSPFArea`

Returns OSPFArea class object

`ospfNeighbors ⇒ Array`

Array with OSPFArea class objects

`ospfRouterId ⇒ String`

OSPF router ID

`platformName ⇒ String`

Platform name reported by NetXMS agent

`physicalContainer ⇒ Object`

Physical container object: Rack NetObj or Chassis Chassis

`physicalContainerId ⇒ Integer`

Physical container object id (Rack or Chassis)

`primaryHostName ⇒ String`

Primary host name

`productCode ⇒ String`

Hardware system property - product code

`productName ⇒ String`

Hardware system property - product name

`productVersion ⇒ String`

Hardware system property - product version

`rack ⇒ NetObj`

Will return Rack object as NetObj if node is added in rack

`rackId ⇒ Integer`

Will return Rack id if node is added in rack

`rackHeight ⇒ Integer`

Object height in rack

`rackPosition ⇒ Integer`

Object position in rack

`runtimeFlags ⇒ Integer`

Bit mask of Node runtime flags, `uint32`.

`serialNumber ⇒ String`

Serial number from hardware system property

`snmpOID ⇒ String`

SNMP object identifier (result of `.1.3.6.1.2.1.1.2.0` request)

`snmpProxy ⇒ Node`

Object that is set as SNMP proxy in object properties or NULL.

`snmpSysContact ⇒ String`

SNMP system contact (result of `.1.3.6.1.2.1.1.4.0` request)

`snmpSysLocation ⇒ String`

SNMP system location (result of `.1.3.6.1.2.1.1.6.0` request)

`snmpSysName ⇒ String`

SNMP system name (result of `.1.3.6.1.2.1.1.5.0` request)

`snmpVersion ⇒ Integer`

Configured SNMP version:

- 0: SNMP version 1
- 1: SNMP version 2c
- 2: SNMP version 3

`softwarePackages ⇒ Array`

Returns array of SoftwarePackage class objects

`sysDescription ⇒ String`

System description (value of `System.Uname` for nodes with agents or `.1.3.6.1.2.1.1.1.0` for SNMP nodes)

`tunnel ⇒ Tunnel`

Tunnel object f this node or NULL if there is no tunnel active and bound to this node.

`vendor ⇒ String`

Hardware vendor information

`vlans ⇒ Array`

Array with object VLAN objects (`null` if there are no VLANs)

`zone ⇒ Zone`

Zone object (`null` if zoning is disabled)

`zoneProxyAssignments ⇒ Integer`

Number of objects where this node is selected as either primary or backup zone proxy (`0` if zoning is disabled or this node is not a zone proxy).

`zoneProxyStatus ⇒ Boolean`

Status of this node as zone proxy (`true` if active).

`zoneUIN ⇒ Integer`

This node zone UIN

Instance methods

`callWebService(webSvcName, requestType, ⋯) ⇒ WebServiceResponse`

Finds web service by name, makes request and returns the WebServiceResponse object.

Parameters

webSvcName	String	Web service name.
requestType	String	Request type. One of: GET, DELETE, PATCH, POST, PUT
data	String or JsonObject	Depending on request type data might be required and provided as a string or JSON object. GET and DELETE requests do not allow data, PATCH, POST, PUT require data.
contentType	String	Optional parameter. Type of provided data that will be set to "Content-Type" header of request. Default type is "application/json".
...	Strings	Optional additional parameter(s) that will be used in web service definition to expand %1, %2... macro.

Return

Instance of WebServiceResponse with request result.

Example

```
//Valid example when request fails because there is no connection to agent
result = $node->callWebService("Web service name", "GET", "additional/ult/path");
println(result->success); // "false"
println(result->errorMessage); // "No connection with agent"
println(result->agentErrorCode); // "0"
println(result->httpResponseCode); // "0"
println(result->document); // ""
```

```
//Successful post request
result = $node2->callWebService("Web service name", "POST", "{ \"id\":10 }",
"application/json", "additional/ult/path");
println(result->success); // "true"
println(result->errorMessage); // ""
println(result->agentErrorCode); // "0"
println(result->httpResponseCode); // "200"
println(result->document); // ""
```

`createSNMPTransport(port, community, context, failIfUnreachable) ⇒ SNMPTransport`

Create SNMP transport object of class SNMPTransport with communication settings defined on the node. It is possible to specify a community string but only community strings listed in Network Credentials will be accepted. Creation of SNMP transport is a preparatory operation enabling subsequent sending of SNMP requests to node. However, creation of SNMP transport does not guarantee that the node is accessible.

Parameters

port	Integer	Optional parameter with port.
community	String	Optional parameter with community string.
context	String	Optional parameter with context.
failIfUnreachable	String	Optional parameter, is provided and is true, check that SNMP is reachable on the node before creating SNMP transport. **This parameter was added in version 4.4.1.**

Return

Instance of SNMPTransport or `null` if failed or node was not reacheable when failIfUnreachable was set to true.

`enable8021xStatusPolling(flag) ⇒ void`

Enable or disable 802.1x port state checking during status polls.

Parameters

flag	Boolean	If 802.1x port state checking should be enabled.

`enableAgent(flag) ⇒ void`

Enable or disable usage of NetXMS agent for all polls.

Parameters

flag	Boolean	If agent usage should be enabled.

`enableDiscoveryPolling(flag) ⇒ void`

Enable or disable discovery polling.

Parameters

flag	Boolean	If discovery polling should be enabled.

`enableEtherNetIP(flag) ⇒ void`

Enable or disable usage of EtherNet/IP for polls.

Parameters

flag	Boolean	If EtherNet/IP should be enabled.

`enableIcmp(flag) ⇒ void`

Enable or disable usage of ICMP pings for status polls.

Parameters

flag	Boolean	If ICMP pings should be enabled.

`enableModbusTcp(flag) ⇒ void`

Enable or disable usage of Modbus TCP for polls.

Parameters

flag	Boolean	If Modbus TCP should be enabled.

`enablePrimaryIPPing(flag) ⇒ void`

Enable or disable usage of ICMP ping for primary IP.

Parameters

flag	Boolean	If primary IP ping should be enabled.

`enableRoutingTablePolling(flag) ⇒ void`

Enable or disable routing table polling.

Parameters

flag	Boolean	If routing table polls should be enabled.

`enableSnmp(flag) ⇒ void`

Enable or disable usage of SNMP for all polls.

Parameters

flag	Boolean	If SNMP communication should be enabled.

`enableSsh(flag) ⇒ void`

Enable or disable usage of SSH

Parameters

flag	Boolean	If SSH communication should be enabled.

`enableTopologyPolling(flag) ⇒ void`

Enable or disable topology polling.

Parameters

flag	Boolean	If topology polls should be enabled.

`enableWinPerfCountersCache(flag) ⇒ void`

Enable or disable reading of Windows performance counters metadata

Parameters

flag	Boolean	If reading of Windows performance counters metadata should be enabled.

`executeAgentCommand(command, ⋯) ⇒ Boolean`

Execute agent command (action) on node.

Parameters

command	String	Command to be executed.
...	String	Optional arguments for command

Return

True if command execution is successful or false otherwise.

`executeAgentCommandWithOutput(command, ⋯) ⇒ String`

Execute agent command (action) on node and return it's output.

Parameters

command	String	Command to be executed.
...	String	Optional arguments for command

Return

String with output of the command or null on failure.

`executeSSHCommand(command) ⇒ array`

Execute SSH command on node.

Parameters

command	String	Command to be executed.

Return

Array of strings with output of the command or null on failure.

`getInterface(ifIdentifier) ⇒ Interface`

Get interface object by index, MAC address or name. If name is number method will assume that it is index.

Parameters

ifIdentifier	Integer or String	Index, MAC address or name of interface.

Return

Interface object.

Example

```
println($node->getInterface("00:14:22:04:25:37")->name); // Will print "wlp4s0"
println($node->getInterface(3)->name); // Will print "wlp4s0"
println($node->getInterface("wlp4s0")->name); // Will print "wlp4s0"
```

`getInterfaceByIndex(ifIndex) ⇒ Interface`

Get interface object by index.

Parameters

ifIndex	Integer	Index of interface.

Return

Interface object.

Example

```
println($node->getInterfaceByIndex(3)->name); // Will print "wlp4s0"
```

`getInterfaceByMACAddress(ifMACAddr) ⇒ Interface`

Get interface object by MAC address.

Parameters

ifMACAddr	String	MAC address of interface.

Return

Interface object.

Example

```
println($node->getInterfaceByMACAddress("00:14:22:04:25:37")->name); // Will print
"wlp4s0"
```

`getInterfaceByName(IfName) ⇒ Interface`

Get interface object by name.

Parameters

IfName	String	Name of interface.

Return

Interface object.

Example

```
println($node->getInterfaceByName("wlp4s0")->name); // Will print "wlp4s0"
```

`getInterfaceName(ifIndex) ⇒ String`

Get interface name by index.

Parameters

ifIndex	Integer	Index of interface.

`getWebService(webSvcName) ⇒ WebService`

Get web service object by name.

Parameters

webSvcName	String	Name of interface.

Return

WebService object.

Example

```
webSvc = $node->getWebService("webSvcName");
```

`readAgentParameter(name) ⇒ String`

Reads current value of agent metric.

Parameters

name	String	Parameter name.

`readAgentList(name) ⇒ Array`

Reads current value of agent list metric and returns array of strings.

Parameters

name	String	List name.

`readAgentTable(name) ⇒ Table`

Reads current value of agent table metric and returns Table.

Parameters

name	String	Table name.

`readDriverParameter(name) ⇒ String`

Request driver-specific metric directly from network device driver (e.g. Rital).

Parameters

name	String	List name.

`readInternalParameter(name) ⇒ String`

Read internal parameter

Parameters

name	String	Parameter name

Return

Value of requested internal parameter

`readInternalTable(name) ⇒ Table`

Read internal table

Parameters

name	String	Table name

Return

Value of requested internal table of class Table

`readWebServiceList(name) ⇒ Array`

Read list from node using web service

Parameters

name	String	Name is expected in form service:path or service(arguments):path

Return

Array with string with web service instances

`readWebServiceParameter(name) ⇒ String`

Read from node web service parameter

Parameters

name	String	Name is expected in form service:path or service(arguments):path

Return

String with result read from web service

`setIfXTableUsageMode(mode) ⇒ void`

Set ifXTable usage mode 0 - use default, 1 - enable, 2 - disable

Parameters

mode	Integer	usage mode

Constants

Node flags

Description	Value
DCF_DISABLE_STATUS_POLL	0x00000001
DCF_DISABLE_CONF_POLL	0x00000002
DCF_DISABLE_DATA_COLLECT	0x00000004
DCF_LOCATION_CHANGE_EVENT	0x00000008
NF_EXTERNAL_GATEWAY	0x00010000
NF_DISABLE_DISCOVERY_POLL	0x00020000
NF_DISABLE_TOPOLOGY_POLL	0x00040000
NF_DISABLE_SNMP	0x00080000
NF_DISABLE_NXCP	0x00100000
NF_DISABLE_ICMP	0x00200000
NF_FORCE_ENCRYPTION	0x00400000
NF_DISABLE_ROUTE_POLL	0x00800000
NF_AGENT_OVER_TUNNEL_ONLY	0x01000000
NF_SNMP_SETTINGS_LOCKED	0x02000000
NF_PING_PRIMARY_IP	0x04000000
NF_DISABLE_ETHERNET_IP	0x08000000
NF_DISABLE_PERF_COUNT	0x10000000

Node runtime flags

Description	Value
DCDF_QUEUED_FOR_STATUS_POLL	0x00000001
DCDF_QUEUED_FOR_CONFIGURATION_POLL	0x00000002
DCDF_QUEUED_FOR_INSTANCE_POLL	0x00000004
DCDF_DELETE_IN_PROGRESS	0x00000008
DCDF_FORCE_STATUS_POLL	0x00000010
DCDF_FORCE_CONFIGURATION_POLL	0x00000020
DCDF_CONFIGURATION_POLL_PASSED	0x00000040
DCDF_CONFIGURATION_POLL_PENDING	0x00000080
NDF_QUEUED_FOR_TOPOLOGY_POLL	0x00010000
NDF_QUEUED_FOR_DISCOVERY_POLL	0x00020000
NDF_QUEUED_FOR_ROUTE_POLL	0x00040000
NDF_RECHECK_CAPABILITIES	0x00080000
NDF_NEW_TUNNEL_BIND	0x00100000

Node capability flags

Value	Description
0x00000001	Node supports SNMP
0x00000002	NetXMS agent detected on the node
0x00000004	Node is network bridge

Value	Description
0x00000008	Node is IP router
0x00000010	Node is management server (NetXMS server itself)
0x00000020	Node is printer
0x00000040	Node supports OSPF
0x00000080	CheckPoint SNMP agent detected on port 260
0x00000100	CDP supported
0x00000200	NDP(SONMP) support detected on the node (Nortel/Synoptics/Bay Networks) topology discovery)
0x00000400	Node supports LLDP
0x00000800	Node supportes VRRP
0x00001000	VLAN information available on the node
0x00002000	802.1x support detected
0x00004000	Spanning Tree (IEEE 802.1d) enabled on node
0x00008000	Node supports ENTITY-MIB
0x00010000	Node supports ifXTable via SNMP (64-bit counters for interface statistics)
0x00020000	Agent supports 64-bit interface counters
0x00040000	Node supports Windows PDH parameters
0x00080000	Node is wireless network controller
0x00100000	Node supports SMCLP protocol
0x00200000	Running agent is upgraded to new policy type
0x00400000	User (support) agent is installed

Node types

Value	Description
0	Unknown
1	Physical
2	Virtual
3	Controller
4	Container

NodeDependency

Detailed information about dependent nodes, usually accessible via instance attribute `dependentNodes` in class Node.

Instance attributes

`id ⇒ Integer`

Node id

`isAgentProxy ⇒ Boolean`

Node is an agent proxy. Helper function for accesing specific bit of `type` field.

`isDataCollectionSource ⇒ Boolean`

Node is a data collection source for another node. Helper function for accesing specific bit of `type` field.

`isICMPProxy ⇒ Boolean`

Node is an ICMP proxy for another node. Helper function for accesing specific bit of `type` field.

`isSNMPProxy ⇒ Boolean`

Node is a SNMP proxy for another node. Helper function for accesing specific bit of `type` field.

`type ⇒ Integer`

Type of of the dependencty this object represent. This field is a bitmask, flags are documented at Type bitmask.

Constants

Type bitmask

Value	Description
0x01	Agent proxy
0x02	SNMP proxy
0x04	ICMP proxy
0x08	Data collection source

Object

Class Object is the root of the class hierarchy. Every class has Object as a superclass.

Instance attributes

`__class ⇒ Class`

Object of Class class

Instance methods

`__get(name) ⇒ Value`

Get any attribute value by it's name

Parameters

name	String	Name of the attribute

Return

Attribute value

`__invoke(name, ⋯) ⇒ Value`

Execute any method, by it's name

Parameters

name	String	Name of the method
…	Value	Any values that are required for method execution

Return

Method return value if method returns something

OSPFArea

OSPF area

Instance attributes

`areaBorderRouterCount ⇒ Integer`

Area border router count

`asBorderRouterCount ⇒ Integer`

Autonomous system border router count

`id ⇒ String`

Area ID

`lsaCount ⇒ Integer`

Link state advertisement count

OSPFNeighbor

OSPF neighbor

Instance attributes

`areaId ⇒ String`

Area ID

`ifIndex ⇒ String`

Interface index

`ipAddress ⇒ String`

IP address

`isVirtual ⇒ Boolean`

`TRUE` if it is virtual

`node ⇒ Node`

Object of Node class

`nodeId ⇒ Integer`

Node ID

`routerId ⇒ String`

OSPF router ID

`state ⇒ Integer`

State

`stateText ⇒ String`

State text

Sensor

Object represent sensor, extends DataCollectionTarget

Instance attributes

`description ⇒ String`

Description

`frameCount ⇒ Integer`

Frame count

`lastContact ⇒ Integer`

Last contact UNIX timestamp

`metaType ⇒ String`

Meta type

`protocol ⇒ Integer`

Communication protocol

`serial ⇒ String`

Serial number

`type ⇒ Integer`

<<sensor-type>

`vendor ⇒ String`

Vendor

Constants

Sensor type

Description	Value
Unknown	1
LoraWAN	2
DLMS	3

SNMPTransport

Represents SNMP Transport functionality. Objects of this class are typically obtained from nodes that support SNMP. Objects of this class used to access SNMP data form node.

Instance attributes

`snmpVersion ⇒ String`

SNMP version used by the transport. Can be "1", "2c" or "3"

Instance methods

`get(oid) ⇒ SNMPVarBind`

Get the object value from specific node with SNMP GET request. The node and all SNMP communication details defined by SNMP transport. Returns SNMPVarBind object. Will return null on failure.

Parameters

oid	String	SNMP object id.

`getValue(oid) ⇒ String`

Get the object value from specific node with SNMP GET request. The node and all SNMP communication details defined by SNMP transport. This function is similar to SNMPGet but returns string instead of an SNMPVarBind object. Will return null on failure.

Parameters

oid	String	SNMP object id.

`getValues(array) ⇒ String`

Get object values from specific node with SNMP GET request. Request will contain all values that are provided in array. The node and all SNMP communication details defined by SNMP transport.

Parameters

array	Array	Array with SNMP object ids to get

Return

Array with resulting strings for each array

`set(oid, value, dataType) ⇒ Boolean`

Assign a specific value to the given SNMP object for the node. The node and all SNMP communication details defined by SNMP transport. Will return TRUE on success, FALSE in case of failure.

Parameters

oid	String	SNMP object id.
value	String	Value to assign to oid.
dataType	String	SNMP data type (optional).

`walk(oid) ⇒ Array`

Get an array of the SNMPVarBind from specific node with SNMP WALK request. The node and all SNMP communication details defined by SNMP transport. Will return null on failure.

Parameters

oid	String	SNMP object id.

Constants

SNMP data types

Description	Value
Integer.	INTEGER
Same as INTEGER.	INT
Octet string.	STRING
Object id.	OID
IP address.	IPADDR
Same as IPADDR.	IP ADDRESS
32-bit counter.	COUNTER32
32-bit unsigned integer.	GAUGE32
Timeticks.	TIMETICKS
64-bit counter.	COUNTER64
32-bit unsigned integer.	UINTEGER32
Same as UINTEGER32.	UINT32

SNMPVarBind

Represents an SNMP varbind concept in NetXMS. A varbind logically consists of an OID and a value.

Instance attributes

`name ⇒ String`

Object name (OID string).

`type ⇒ Integer`

ASN.1 type.

`value ⇒ String`

Object value as a string.

`printableValue ⇒ String`

Object value as a string with non-printable characters replaced by `?` character.

`valueAsIp ⇒ String`

Object value IP address, represented as string.

`valueAsMac ⇒ String`

Object value as MAC address, represented as string.

Methods

`getValueAsByteStream() ⇒ ByteStream`

Returns object value as ByteStream.

`getValueAsString(codepage) ⇒ String`

Constructs a new String by decoding the varbind value using the specified codepage.

SoftwarePackage

Represents software package installed on Node object.

Instance attributes

`changeCode ⇒ Integer`

Change code one of next:

- CHANGE_NONE: 0
- CHANGE_ADDED: 1
- CHANGE_UPDATED: 2

`date ⇒ String`

Change date in YYYYMMDD format

`description ⇒ String`

PAckage description

`name ⇒ String`

Package name

`timestamp ⇒ Integer`

Package change date as UNIX timestamp

`url ⇒ String`

PAckage URL

`vendor ⇒ String`

PAckage vendor

`version ⇒ String`

Package version

Subnet

Object represent subnet, extends NetObj

Instance attributes

`ipNetMask ⇒ Integer`

Subnet mask

`isSyntheticMask ⇒ Boolean`

`TRUE` is mask is synthetic

`zone ⇒ Zone`

Zone object (`null` if zoning is disabled)

`zoneUIN ⇒ Integer`

This subnet zone UIN

Table

Represents table object (usually it's value of table DCI).

Instance attributes

`columnCount ⇒ Number`

Number of columns.

`columns ⇒ Array<TableColumn>`

Array of column definitions.

`instanceColumns ⇒ Array<TableColumn>`

Array of column definitions including only columns marked as instance columns.

`instanceColumnIndexes ⇒ Array<Integer>`

Array with indexes of columns marked as instance columns.

`rowCount ⇒ Numbers`

Number of rows.

`rows ⇒ Array<TableRow>`

Array of rows with data.

`title ⇒ String`

Title of table.

Instance methods

`addColumn(name, [type], [displayName], [isInstance]) ⇒ Integer`

Adds column to table

Parameters

name	String	Column name
type	Integer	Data type, optional parameter. Default is String
displayName	String	Column display name
isInstance	Boolean	`True` if column is instance column

Return

Column index

`addRow() ⇒ Integer`

Adds row to table

Return

Row index

`deleteColumn(columnId) ⇒ void`

Delete column

Parameters

columnId	Integer	Column index

`deleteRow(rowId) ⇒ void`

Delete row

Parameters

rowId	Integer	Row index

`findRowByInstance(instance, instance2, ⋯) ⇒ TableRow`

Finds row by instance

Parameters

instance	String	Instance string
instance2	String	Optional. If instance is composed of several columns, these can be supplied as additional parameters.

Return

Table row that corresponds to provided instance

`findRowIndexByInstance(instance, instance2, ⋯) ⇒ Integer`

Finds row index by instance

Parameters

instance	String	Instance string
instance2	String	Optional. If instance is composed of several columns, these can be supplied as additional parameters.

Return

Table row index corresponds to provided instance

`get(rowId, columnId) ⇒ String`

Get cell value by row and column id

Parameters

rowId	Integer	Row index
columnId	Integer	Column index

Return

Cell value as string

`getColumnIndex(columnName) ⇒ Integer`

Get column index by column name

Parameters

columnName	String	Column name

Return

Column name

`getColumnName(columnId) ⇒ String`

Get column name by column index

Parameters

columnName	Integer	Column name

Return

Column index

`set(rowId, columnId, value) ⇒ void`

Set column value by row and column index

Parameters

rowId	Integer	Row index
columnId	Integer	Column index
value	String	New value

Return

Column index

Constructors

`Table()`

Creates new Table object.

TableColumn

Represents table column definition object (used by Table class).

Instance attributes

`dataType ⇒ Integer`

Data type

`displayName ⇒ String`

Display name

`isInstanceColumn ⇒ Boolean`

`TRUE` if column is marked as instance column

`name ⇒ String`

Column name

TableRow

Represents table row definition object (used by Table class).

Instance attributes

`index ⇒ Integer`

Row index number

`instance ⇒ String`

Row instance name

`values ⇒ Array<String>`

Row values for all columns

Instance methods

`get(columnId) ⇒ String`

Get cell value

Parameters

columnId	Integer	Column id

Return

Cell value

`set(columnId, value) ⇒ void`

Set cell value

Parameters

columnId	Integer	Column id
value	Integer	New cell value

Template

Object represent template, extends NetObj.

Instance attributes

`autoApplyScript ⇒ String`

Source of the script for automatic binding.

`isAutoApplyEnabled ⇒ Boolean`

Indicate if automatic binding is enabled.

`isAutoRemoveEnabled ⇒ Boolean`

Indicate if automatic unbinding is enabled.

`version ⇒ Integer`

Template version

Instance methods

`applyTo(object) ⇒ void`

Apply teplate on object

Parameters

object	Object	Object this template to be applied on

`removeFrom(object) ⇒ void`

emplate form object

Parameters

object	Object	Object this template to be removed form

`setAutoApplyMode(enableBind, enableUnbind) ⇒ void`

Set automatic bind mode for the container.

Parameters

enableBind	Boolean	Script should be used for automatic binding.
enableUnbind	Boolean	Script should be used for automatic unbinding.

`setAutoApplyScript(script) ⇒ void`

Update automatic binding script source.

Parameters

script	String	Script content.

TIME

Class containing a calendar date and time broken down into its components. For convenience, all attributes has aliases to match struct tm provided in libc.

Instance attributes

`sec ⇒ Number`

`tm_sec ⇒ Number`

Seconds after the minute.

`min ⇒ Number`

`tm_min ⇒ Number`

Minutes after the hour.

`hour ⇒ Number`

`tm_hour ⇒ Number`

Hours since midnight.

`mday ⇒ Number`

`tm_mday ⇒ Number`

Day of the month.

`mon ⇒ Number`

`tm_mon ⇒ Number`

Months since January.

`year ⇒ Number`

`tm_year ⇒ Number`

Year.

`yday ⇒ Number`

`tm_yday ⇒ Number`

Days since January 1.

`wday ⇒ Number`

`tm_wday ⇒ Number`

Days since Sunday.

`isdst ⇒ Boolean`

`tm_isdst ⇒ Boolean`

Daylight Saving Time flag.

Constructors

`TIME()`

Creates new TIME object.

Tunnel

Object contains information about tunnel.

Instance attributes

`address ⇒ InetAddress`

InetAddress class object with address of the node that opened this tunnel

`agentBuildTag ⇒ String`

Agent build tag

`agentId ⇒ String`

Agent ID of the agent that opened the tunnel

`agentVersion ⇒ String`

Agent version of the node that opened the tunnel

`certificateExpirationTime ⇒ Number`

UNIX timestamp with current certificate expiration day

`guid ⇒ String`

Tunnel GUID

`hostname ⇒ String`

Hostname of the node that opened the tunnel

`hardwareId ⇒ String`

Hardware ID of the node that opened the tunnel

`id ⇒ Number`

Tunnel id

`isAgentProxy ⇒ Boolean`

`TRUE` if agent proxy is enabled on the agent that opened the tunnel

`isBound ⇒ Boolean`

`TRUE` if tunnel is bound

`isSnmpProxy ⇒ Boolean`

`TRUE` if SNMP proxy is enabled on the agent that opened the tunnel

`isSnmpTrapProxy ⇒ Boolean`

`TRUE` if SNMP trap proxy is enabled on the agent that opened the tunnel

`isUserAgentInstalled ⇒ Boolean`

`TRUE` if user agent is installed on the agent that opened the tunnel

`macAddresses ⇒ Array`

Array with MAC addresses as Strings

`platformName ⇒ String`

Platform name of the node that opened the tunnel

`startTime ⇒ Number`

UNIX timestamp with tunnel start time

`serialNumber ⇒ String`

Serial number

`systemInfo ⇒ String`

System information of the node that opened the tunnel

`systemName ⇒ String`

System name of the node that opened the tunnel

`zoneUIN ⇒ Number`

Node's zone UIN if set in agent's configuration file

User

Object represent user object and extends UserDBObject.

Instance attributes

`authMethod ⇒ Integer`

Authentication method.

`certMappingData ⇒ String`

Data that will be used to mpa certificate

`certMappingMethod ⇒ Integer`

Certificate mapping methods.

`disabledUntil ⇒ String`

UNIX timestamp with date until this user is disabled

`disabledUntil ⇒ String`

UNIX timestamp with date until this user is disabled

`email ⇒ String`

Email set in user properties

`fullName ⇒ String`

User full name

`graceLogins ⇒ Integer`

Grace login count

`phoneNumber ⇒ String`

Phone number set in user properties

`lastLogin ⇒ Integer`

UNIX timestamp with last user's login time

`xmppId ⇒ String`

XMPP id

Constants

Status propagation types

Code	Description
0	Password

Code	Description
1	Radius
2	Certificate
3	Certificate or password
4	Certificate or radius

Status propagation types

Code	Description
0	Subject
1	Public key
2	CN

UserDBObject

Base class for User and UserGroup with common fields

Instance attributes

`description ⇒ String`

Description

`flags ⇒ Integer`

Flags

`guid ⇒ String`

GUID

`id ⇒ Integer`

Id

`isDeleted ⇒ Boolean`

`TRUE` if user DB object is deleted

`isDisabled ⇒ Boolean`

`TRUE` if user DB object is disabled

`isGroup ⇒ Boolean`

`TRUE` if user DB object is group object: UserGroup

`isModified ⇒ Boolean`

`TRUE` if user DB object is modified

`isLDAPUser ⇒ Boolean`

`TRUE` if user DB object is synchronized from LDAP

`ldapDN ⇒ String`

Get user DB object LDAP domain name

`ldapId ⇒ String`

Get user DB object LDAP id (value depends on the field that is ues as LDAP object id)

`name ⇒ String`

Object name

`systemRights ⇒ Integer64`

Field with system rights as bit flags.

Constants

Status propagation types

Code	Description
0x000000000001	Manage users
0x000000000002	Change server configuration
0x000000000004	Configure traps
0x000000000008	SYSTEM_ACCESS_MANAGE_SESSIONS
0x000000000010	SYSTEM_ACCESS_VIEW_EVENT_DB
0x000000000020	SYSTEM_ACCESS_EDIT_EVENT_DB
0x000000000040	SYSTEM_ACCESS_EPP
0x000000000080	SYSTEM_ACCESS_MANAGE_ACTIONS
0x000000000100	SYSTEM_ACCESS_DELETE_ALARMS
0x000000000200	SYSTEM_ACCESS_MANAGE_PACKAGES
0x000000000400	SYSTEM_ACCESS_VIEW_EVENT_LOG
0x000000000800	SYSTEM_ACCESS_MANAGE_TOOLS
0x000000001000	SYSTEM_ACCESS_MANAGE_SCRIPTS
0x000000002000	SYSTEM_ACCESS_VIEW_TRAP_LOG
0x000000004000	SYSTEM_ACCESS_VIEW_AUDIT_LOG
0x000000008000	SYSTEM_ACCESS_MANAGE_AGENT_CFG
0x000000010000	SYSTEM_ACCESS_PERSISTENT_STORAGE
0x000000020000	SYSTEM_ACCESS_SEND_NOTIFICATION
0x000000040000	SYSTEM_ACCESS_MOBILE_DEVICE_LOGIN
0x000000080000	SYSTEM_ACCESS_REGISTER_AGENTS
0x000000100000	SYSTEM_ACCESS_READ_SERVER_FILES
0x000000200000	SYSTEM_ACCESS_SERVER_CONSOLE

Code	Description
0x000000400000	SYSTEM_ACCESS_MANAGE_SERVER_FILES
0x000000800000	SYSTEM_ACCESS_MANAGE_MAPPING_TBLS
0x000001000000	SYSTEM_ACCESS_MANAGE_SUMMARY_TBLS
0x000002000000	SYSTEM_ACCESS_REPORTING_SERVER
0x000004000000	SYSTEM_ACCESS_XMPP_COMMANDS
0x000008000000	SYSTEM_ACCESS_MANAGE_IMAGE_LIB
0x000010000000	SYSTEM_ACCESS_UNLINK_ISSUES
0x000020000000	SYSTEM_ACCESS_VIEW_SYSLOG
0x000040000000	SYSTEM_ACCESS_USER_SCHEDULED_TASKS
0x000080000000	SYSTEM_ACCESS_OWN_SCHEDULED_TASKS
0x000100000000	SYSTEM_ACCESS_ALL_SCHEDULED_TASKS
0x000200000000	SYSTEM_ACCESS_SCHEDULE_SCRIPT
0x000400000000	SYSTEM_ACCESS_SCHEDULE_FILE_UPLOAD
0x000800000000	SYSTEM_ACCESS_SCHEDULE_MAINTENANCE
0x001000000000	SYSTEM_ACCESS_MANAGE_REPOSITORIES
0x002000000000	SYSTEM_ACCESS_VIEW_REPOSITORIES
0x004000000000	SYSTEM_ACCESS_VIEW_ALL_ALARMS
0x008000000000	SYSTEM_ACCESS_EXTERNAL_INTEGRATION
0x010000000000	SYSTEM_ACCESS_SETUP_TCP_PROXY
0x020000000000	SYSTEM_ACCESS_IMPORT_CONFIGURATION
0x040000000000	SYSTEM_ACCESS_UA_NOTIFICATIONS
0x080000000000	SYSTEM_ACCESS_WEB_SERVICE_DEFINITIONS

UserGroup

Object represent user group object and extends UserDBObject.

Instance attributes

`memberCount ⇒ Integer`

Group member count

`members ⇒ Array`

Array with objects: User or UserGroup

VLAN

Represents VLAN object.

Instance attributes

`id ⇒ Integer`

VLAN id.

`name ⇒ String`

VLAN name.

`interfaces ⇒ Array`

Interfaces in that VLAN (array of objects of class Interface).

WebService

An object that represents a combination of node and web service definition, that will be used for requests.

Instance attributes

`id ⇒ Integer`

Web service id

`name ⇒ String`

Web service name

`description ⇒ String`

Web service description

Instance methods

`get(…) ⇒ WebServiceResponse`

Execute GET request from agent on associated node. Associated node is the node on which getWebService method was executed. This node is used for macro expansion.

Parameters

…	Strings	Optional additional parameter(s) that will be used in web service definition to expand %1, %2… macro.

Return

Instance of WebServiceResponse with request result.

Example

```
webSvc = $node->getWebService("webSvcName");
```

```
result = webSvc->get();
println(result->document); // will print result

result = webSvc->get("additional", "parameters");
println(result->document); // will print result
```

`delete(…) ⇒ WebServiceResponse`

Execute DELETE request from agent on associated node. Associated node is the node on which getWebService method was executed. This node is used for macro expansion.

Parameters

…	Strings	Optional additional parameter(s) that will be used in web service definition to expand %1, %2… macro.

Return

Instance of WebServiceResponse with request result.

Example

```
webSvc = $node->getWebService("webSvcName");
result = webSvc->delete();
println(result->success); //will print "true" if request was successful or "false"
otherwise

result = webSvc->delete("additional", "parameters");
println(result->success); //will print "true" if request was successful or "false"
otherwise
```

`patch(data, contentType, …) ⇒ WebServiceResponse`

Execute PATCH request from agent on associated node. Associated node is the node on which getWebService method was executed. This node is used for macro expansion.

Parameters

data	String or JsonObject	Data that will be set to the web service.
contentType	String	Optional parameter. Type of provided data that will be set to "Content-Type" header of request. Default type is "application/json".
…	Strings	Optional additional parameter(s) that will be used in web service definition to expand %1, %2… macro.

Return

Instance of WebServiceResponse with request result.

Example

```
webSvc = $node->getWebService("webSvcName");
```

```
result = webSvc->patch("{ \"id\":10 }");
println(result->success); //will print "true" if request was successful or "false"
otherwise
to
json = new JsonObject();
json->set("id", 42);
result = webSvc->patch(json,"application/json", "additional", "parameters");
println(result->success); //will print "true" if request was successful or "false"
otherwise
```

`post(data, contentType, ⋯) ⇒ WebServiceResponse`

Execute POST request from agent on associated node. Associated node is the node on which getWebService method was executed. This node is used for macro expansion.

Parameters

data	String or JsonObject	Data that will be set to the web service.
contentType	String	Optional parameter. Type of provided data that will be set to "Content-Type" header of request. Default type is "application/json".
...	Strings	Optional additional parameter(s) that will be used in web service definition to expand %1, %2... macro.

Return

Instance of WebServiceResponse with request result.

Example

```
webSvc = $node->getWebService("webSvcName");

result = webSvc->post("{ \"id\":10 }");
println(result->success);//will print "true" if request was successful or "false"
otherwise

json = new JsonObject();
json->set("id", 42);
result = webSvc->post(json,"application/json", "additional", "parameters");
println(result->success); //will print "true" if request was successful or "false"
otherwise
```

`put(data, contentType, ⋯) ⇒ WebServiceResponse`

Execute PUT request from agent on associated node. Associated node is the node on which getWebService method was executed. This node is used for macro expansion.

Parameters

data	String or JsonObject	Data that will be set to the web service.
contentType	String	Optional parameter. Type of provided data that will be set to "Content-Type" header of request. Default type is "application/json".
...	Strings	Optional additional parameter(s) that will be used in web service definition to expand %1, %2... macro.

Return

Instance of WebServiceResponse with request result.

Example

```
webSvc = $node->getWebService("webSvcName");

result = webSvc->put("{ \"id\":10 }");
println(result->success); //will print "true" if request was successful or "false"
otherwise

json = new JsonObject();
json->set("id", 42);
result = webSvc->put(json,"application/json", "additional", "parameters");
println(result->success); //will print "true" if request was successful or "false"
otherwise
```

WebServiceResponse

Contains all information about web service custom request execution result

Instance attributes

`agentErrorCode ⇒ Integer`

Agent error code

`document ⇒ String`

Document that was returned by web service as a response to the request

`errorMessage ⇒ String`

Human readable error message set by server or agent

`httpResponseCode ⇒ Integer`

HTTP response code

`success ⇒ Boolean`

Will return `True` in case of success and `False` if there was problem on server or on agent

Zone

Represent network zone. Inherit all attributes and methods of the NetObj class.

Instance attributes

Previously available attributes proxyNode and proxyNodeId were deprecated starting from version 3.0.

`proxyNodes ⇒ Array<Node>`

Array of Node objects that are currently set as proxies for this zone.

`proxyNodeIds ⇒ Array<Integer>`

Array of integers representing identifiers of node objects that are currently set as proxies for this zone.

`uin ⇒ Integer`

Zone UIN (Unique Identification Number).

Global Constants

Status of the BusinessService class object

Constant	Value	Description
BusinessServiceState::OPERATIONAL	0	No failed checks
BusinessServiceState::DEGRADED	2	One of checks returned not normal, but is not failed yet
BusinessServiceState::FAILED	4	At least one of checks failed

Type of BusinessServiceCheck class

Constant	Value	Description
BusinessServiceType::NONE	0	None
BusinessServiceType::SCRIPT	1	Script
BusinessServiceType::DCI	2	DCI
BusinessServiceType::OBJECT	3	Object

Data types of the DCI class

Constant	Value	Description
DCI::INT32	0	Signed 32 bit integer
DCI::UINT32	1	Unsigned 32 bit integer
DCI::INT64	2	Signed 64 bit integer
DCI::UINT64	3	Unsigned 64 bit integer
DCI::STRING	4	String
DCI::FLOAT	5	Floating point number
DCI::NULL	6	Used internally; should not be used in the scripts
DCI::COUNTER32	7	32 bit counter
DCI::COUNTER64	8	64 bit counter

DCI states of the DCI class

Constant	Value	Description
DCI::ACTIVE	0	Active
DCI::DISABLED	1	Disabled
DCI::UNSUPPORTED	2	Unsupported

DCI data source (origin) of the DCI class

Constant	Value	Description
DataSource::INTERNAL	0	Internal
DataSource::AGENT	1	Agent
DataSource::SNMP	2	SNMP
DataSource::WEB_SERVICE	3	Web service
DataSource::PUSH	4	Push data
DataSource::WINPERF	5	Windows Performance Counters
DataSource::SMCLP	6	SMCLP
DataSource::SCRIPT	7	Script
DataSource::SSH	8	SSH
DataSource::MQTT	9	MQTT
DataSource::DEVICE_DRIVER	10	Network Device driver

Flags for DCI class

Constant	Value	Description
DCI::ALL_THRESHOLDS	0x00002	Process all thresholds
DCI::RAW_VALUE_OCTET_STRING	0x00004	Raw value is octet string
DCI::SHOW_ON_OBJECT_TOOLTIP	0x00008	Show DCI on object tooltip

Constant	Value	Description
DCI::AGGREGATE_ON_CLUSTER	0x00080	Aggregate data on cluster
DCI::TRANSFORM_AGGREGATED	0x00100	Transform agregated data
DCI::CALCULATE_NODE_STATUS	0x00400	Used in node status calculation
DCI::SHOW_IN_OBJECT_OVERVIEW	0x00800	Show in object overview
DCI::AGGREGATE_WITH_ERRORS	0x04000	Aggregate with errors
DCI::HIDE_ON_LAST_VALUES_PAGE	0x08000	Hide on last values page
DCI::STORE_CHANGES_ONLY	0x40000	Stor changed values only

Origin of Event class

Constant	Value	Description
EventOrigin::SYSTEM	0	System
EventOrigin::AGENT	1	Agent
EventOrigin::CLIENT	2	Client
EventOrigin::SYSLOG	3	Syslog
EventOrigin::SNMP	4	SNMP
EventOrigin::NXSL	5	NXSL
EventOrigin::REMOTE_SERVER	6	Remote server
EventOrigin::WINDOWS_EVENT	7	Windows event

Category of HardwareComponent class

Constant	Value	Description
HardwareComponentCategory::OTHER	0	Other
HardwareComponentCategory::BASEBOARD	1	Baseboard
HardwareComponentCategory::PROCESSOR	2	Processor
HardwareComponentCategory::MEMORY	3	Memory
HardwareComponentCategory::STORAGE	4	Storage
HardwareComponentCategory::BATTERY	5	Battery
HardwareComponentCategory::NETWORK_ADAPTER	6	Network adapter

Expected state for Interface class

Constant	Value	Description
InterfaceExpectedState::UP	0	Up
InterfaceExpectedState::DOWN	1	Down
InterfaceExpectedState::IGNORE	2	Ignore

Node state

Constant	Value	Description
NodeState::Unreachable	0x00000001	Node is unreachable
NodeState::NetworkPathProblem	0x00000002	Network path problem

Constant	Value	Description
NodeState::AgentUnreachable	0x00010000	NetXMS agent unreachable
NodeState::SNMPUnreachable	0x00020000	Unreachable via SNMP
NodeState::EtherNetIPUnreachable	0x00040000	Unreachable via EtherNet/IP industrial protocol
NodeState::CacheModeNotSupported	0x00080000	Cache mode is not supported on NetXMS agent
NodeState::SNMPTrapFlood	0x00100000	SNMP trap flood detected
NodeState::ICMPUnreachable	0x00200000	Unreachable via ICMP
NodeState::SSHUnreachable	0x00400000	Unreachable via SSH

Object status codes

Constant	Value	Description
Status::NORMAL	0	Normal
Status::WARNING	1	Warning
Status::MINOR	2	Minor
Status::MAJOR	3	Major
Status::CRITICAL	4	Critical
Status::UNKNOWN	5	Unknown
Status::UNMANAGED	6	Unmanaged
Status::DISABLED	7	Disabled
Status::TESTING	8	Testing

Cluster state

Constant	Value	Description
ClusterState::Unreachable	0x00000001	Unreachable
ClusterState::NetworkPathProblem	0x00000002	Network Path Problem

Constant	Value	Description
ClusterState::Down	0x00010000	Down

Sensor state

Constant	Value	Description
SensorState::Unreachable	0x00000001	Unreachable
SensorState::NetworkPathProblem	0x00000002	Network Path Problem
SensorState::Provisioned	0x00010000	Provisioned
SensorState::Registered	0x00020000	Registered
SensorState::Active	0x00040000	Active
SensorState::PendingConfigUpdate	0x00080000	Pending Config Update

Severity constants

Constant	Value	Description
Severity::NORMAL	0	Normal
Severity::WARNING	1	Warning
Severity::MINOR	2	Minor
Severity::MAJOR	3	Major
Severity::CRITICAL	4	Critical

Status colors

Constant	Value	Description
StatusColor::NORMAL	0	Normal
StatusColor::WARNING	1	Warning

Constant	Value	Description
StatusColor::MINOR	2	Minor
StatusColor::MAJOR	3	Major
StatusColor::CRITICAL	4	Critical
StatusColor::UNKNOWN	5	Critical
StatusColor::UNMANAGED	6	Critical
StatusColor::DISABLED	7	Critical
StatusColor::TESTING	8	Critical

Change Code

Node attributes change code

Constant	Value
ChangeCode::NONE	0
ChangeCode::ADDED	1
ChangeCode::UPDATED	2
ChangeCode::REMOVED	3

Other constants

NXSL::BuildTag

Current server build tag

NXSL::Classes

Array containing all NXSL classes

NXSL::Functions

Array containing names of all NXSL functions

NXSL::SystemIsBigEndian

`TRUE` if system is big endian otherwise `FALSE`

NXSL::Version

Current server version

Formal Grammar

Grammar

```
script ::=
  module |
  expression

module ::=
  module_component { module_component }

module_component ::=
  function |
  statement_or_block |
  use_statement

use_statement ::=
  use any_identifier ";"

any_identifier ::=
  IDENTIFIER |
  COMPOUND_IDENTFIER

function ::=
  function IDENTIFIER "(" [ identifier_list ] ")" block

identifier_list ::=
  IDENTIFIER { "," IDENTIFIER }

block ::=
  "{" { statement_or_block } "}"

statement_or_block ::=
  statement |
  block

statement ::=
  expression ";" |
  builtin_statement |
  ";"

builtin_statement ::=
  simple_statement ";" |
  if_statement |
  do_statement |
  while_statement |
  for_statement |
  foreach_statement |
  switch_statement |
  array_statement |
```

```
  global_statement |
  break ";"
  continue ";"

simple_statement ::=
  keyword [ expression ]

keyword ::=
  exit |
  print |
  println |
  return

if_statement ::=
  if "(" expression ")" statement_or_block [ else statement_or_block ]

for_statement ::=
  for "(" expression ";" expression ";" expression ")" statement_or_block

foreach_statement ::=
  foreach "(" IDENTIFIER ":" expression ")" statement_or_block

while_statement ::=
  while "(" expression ")" statement_or_block

do_statement ::=
  do statement_or_block while "(" expression ")" ";"

switch_statement ::=
  switch "(" expression ")" "{" case { case } [ default ] "}"

case ::=
  case constant ":" { statement_or_block }

default ::=
  default ":" { statement_or_block }

array_statement ::=
  [ global ] array identifier_list ";"

global_statement ::=
  global global_variable_declaration { "," global_variable_declaration } ";"

global_variable_declaration ::=
  IDENTIFIER [ "=" expression ]

expression ::=
  "(" expression ")" |
  IDENTIFIER "=" expression |
  expression "->" IDENTIFIER |
  "-" expression |
```

```
  "!" expression |
  "~" expression |
  inc IDENTIFIER |
  dec IDENTIFIER |
  IDENTIFIER inc |
  IDENTIFIER dec |
  expression "+" expression |
  expression "-" expression |
  expression "*" expression |
  expression "/" expression |
  expression "%" expression |
  expression like expression |
  expression ilike expression |
  expression "~=" expression |
  expression match expression |
  expression imatch expression |
  expression "==" expression |
  expression "!=" expression |
  expression "<" expression |
  expression "<=" expression |
  expression ">" expression |
  expression ">=" expression |
  expression "&" expression |
  expression "|" expression |
  expression "^" expression |
  expression "&&" expression |
  expression "||" expression |
  expression "<<" expression |
  expression ">>" expression |
  expression "." expression |
  expression "?" expression ":" expression |
  operand

operand ::=
  function_call |
  type_cast |
  constant |
  IDENTIFIER

type_cast ::=
  builtin_type "(" expression ")"

builtin_type ::=
  int32 |
  int64 |
  uint32 |
  uint64 |
  real |
  string

function_call ::=
```

```
  IDENTIFIER "(" [ expression { "," expression } ] ")"

constant ::=
  STRING |
  INT32 |
  INT64 |
  UINT32 |
  UINT64 |
  REAL |
  NULL
```

Terminal symbols

```
IDENTIFIER ::= [A-Za-z_\$][A-Za-z_\$0-9]*
COMPOUND_IDENTIFIER ::= { IDENTIFIER}(::{ IDENTIFIER})+
INTEGER ::= \-?(0x)?[0-9]+
INT32 ::= INTEGER
INT64 ::= {INTEGER}L
UINT32 ::= {INTEGER}U
UINT64 ::= {INTEGER}(UL|LU)
REAL ::= \-?[0-9]+\.[0-9]+
```

Examples

Some real life examples.

Small utility scripts

UNIX timestamp to human readable

```
return strftime("%d.%m.%Y %H:%M:%S", $1);
```

Table DCI manipulation script

Get table item. Is used to make DCI from Table DCI.

```
// Warning: this script works only on the same node
//
// $1 - Description
// $2 - column name
table = GetDCIValueByDescription($node, $1);
if (table != NULL) {
    col = table->getColumnIndex($2);
    if (col >= 0) {
        return table->get(0, col);
    }
}
return 0;
```

Primary mac address

Script to print primary MAC address.

```
for(i : $node->interfaces)
{
   for(a : i->ipAddressList)
   {
      if (a->address == $node->ipAddr)
         println(i->macAddr);
   }
}
```

Check if node is under cluster or container

Script that returns `TRUE` if current node is under at least one container or cluster.

```
for (p : $node->parents) {
    if (p->type == 5 or p->type == 14) { //5 is container and 14 is cluster
        return true;
    }
}
return false;
```

Change expected state for all interfaces

Set ignore expected state for all interfaces of all nodes.

```
for (n : GetAllNodes()) {
    println(n->name .. "(" .. n->id .. ")");
    for (i : n->interfaces) {
        println("\t" .. i->name);
        i->setExpectedState("IGNORE");
    }
}
```

Instance filtering script for "Net.InterfaceNames"

Requirements

Instance filtering script for "Net.InterfaceNames" list, that will filter only "eth*" and "bond*" interfaces, that can be used in Net.Interface.BytesIn64 or Net.Interface.BytesOut64 DCIs.

Solution

Select "Agent List" as instance discovery method. Set "Net.InterfaceNames" as list name and add script to instance discovery filter script section:

```
name=$1;
if (name ~= "eth|bond")
{
    return %(name);
}
return false;
```

Filter out some interfaces from creation

Requirements

Filter interfaces that should not be monitored by NetXMS. In this case "isatap*" interfaces.

Solution

Update "Hook:CreateInterface" script in script library.

```
if ( $1->name ~= "^isatap" )
    return false;

return true;
```

Additional Information About Connected Node

Requirements

Add information about name, IP address, and MAC address about connected node to notification about switch port being down and up.

Solution

Script that is being called from Event Processing Policy that adds named event parameter to the event. Parameter is named `additionalInfo`, correspondingly `%<additionalInfo>` macro can be used to add value of that parameter to notification text.

```
// only for interface up and down events
if (($event->name != "SYS_IF_DOWN") && ($event->name != "SYS_IF_UP"))
    return true;

// get interface object from interface index
iface = $node->getInterface($5);
if (iface == null)
    return true;

// get peer node (node connected to this interface) object
peer = iface->peerNode;
if (peer == null)
    return true;

// get peer interface object (needed to obtain MAC address)
peerIface = iface->peerInterface;
if (peerIface != null)
{
    macAddr = peerIface->macAddr;
}
else
{
    macAddr = "<MAC unknown>";
}

// set event's named parameter
```

```
SetEventParameter($event, "additionalInfo",
    "Peer: " .. peer->name .. " " .. peer->ipAddr .. " " .. macAddr);

return true;
```

Enumerate All Nodes

Requirements

Enumerate all nodes in NetXMS database.

Solution 1

Create script in script library which will find "Entire Networks" object and walk down the tree. This script can be executes as an action from event processing policy, or directly from server debug console via exec command or on any node.

In order to be able to access info about all nodes, the CheckTrustedNodes server configuration variable needs to be set to 0.

```
// This function walks object tree recursively starting from given root
function EnumerateNodes(obj, level)
{
    foreach(o : obj->children) {
        for (i = 0; i < level; i++) { print("  "); }
        println("[" .. o->type .. " / " .. classof(o) .. "] " .. o->name);

        EnumerateNodes(o, level + 1);
    }
}
// Find "Entire Network" object and start enumeration from it
EnumerateNodes(FindObject("Entire Network"), 0);
```

Solutions 2

When only nodes are required, not walk down the tree then this script can be used:

```
for (n : GetAllNodes()) {
  println(n->name);
}
```

Enumerate All Custom Attributes for Node

Requirements

Enumerate all custom attributes on a node.

Solution

```
attributes = $node->customAttributes;
foreach(a : attributes->keys)
{
        println(a .. "=" .. attributes[a]);
}
```

Bubble sort with alphabetical sorting

```
function BubbleSort(a)
{
   n = a->maxIndex + 1;
   do
   {
      newn = 0;
      for(i = 1; i < n; i++)
      {
         if (a[i - 1]->compareTo(a[i]) > 0)
         {
            t = a[i - 1];
            a[i - 1] = a[i];
            a[i] = t;
            newn = i;
         }
      }
      n = newn;
   }
   while(n > 1);
   return a;
}
```

Bubble sort with numeric sorting

```
function BubbleSort(a)
{
   n = a->maxIndex + 1;
   do
   {
      newn = 0;
      for(i = 1; i < n; i++)
      {
```

```
         if (a[i - 1] > a[i])
         {
            t = a[i - 1];
            a[i - 1] = a[i];
            a[i] = t;
            newn = i;
         }
      }
      n = newn;
   }
   while(n > 1);
   return a;
}
```

Aggregation of DCI values and applying the 95% percentile average

The example is based around a template which configures ICMP Packet Loss probes. This script will loop around the nodes, collect the required DCI values. The values are then ordered and the top 5 percent discarded, the remaining entries are averaged to give the required value;

```
function main()
{

trace(1, "Global Ping Loss 95");
array pValue;
arrayI  = 0;

foreach(parent : $node->parents)
{
    trace(3, "Parent object: name='" .. parent->name ."' id=" .. parent->id);
    if (parent->name == "all voice")
    {
        foreach(vNode : parent->children)
        {
            dciName = "ICMP: Packet loss to ".vNode->name;
            dciId = FindDCIByDescription(vNode, dciName);
            if (dciId > 0)
            {
                tmpValue = GetDCIValue(vNode,dciId);
                if (tmpValue != null)
                {
                    pValue[arrayI++] = tmpValue;
                }
            }
        }
    }
}
```

```
// Sort the Array
bubbleSort(pValue);

// Apply the 95 percent rule
upTo = arrayI * 0.95;
pLoss = 0;
pCount = 0;
for(ia = 0; ia < upTo; ia++)
{
    pLoss += pValue[ia];
    pCount = ia;
}
p95AvgLoss = pLoss / pCount;

trace(1, "Global Ping Loss 95 Summary: arrayI=".arrayI." upTo=".upTo." p95AvgLoss="
.p95AvgLoss );

return p95AvgLoss;
}

function bubbleSort(arr)
{
    swapped = true;
    while (swapped == true){
        swapped = false;
        for(ia = 1; arr[ia] != null; ia++)
        {
            ib = ia - 1;

            if (arr[ib] > arr[ia]){
                trace(3,"swap: ".ib.":".arr[ib]." with ".ia.":".arr[ia]);
                swapped=true;
                t = arr[ib];
                arr[ib] = arr[ia];
                arr[ia] = t;
                swapped = true;
            }
        }
    }
}

function printArray(arr)
{
    for(ia = 0; arr[ia] != null; ia++)
    {
        trace(1,"printArray: ".ia.":".arr[ia]);
    }
}
```

Read SNMP Value From Node

This script can be put into Script Library and run from server's debug console. It accepts node object name or ID as parameter and prints value of SNMP sysDescription to console.

```
if ($1 == null)
{
   println("Please specify node name as parameter");
   return 3;
}

transport = FindObject($1)->createSNMPTransport();    // Create SNMP transport for
node
if (transport == null)
{
    println("Failed to create SNMP transport, exit");
    return 1;
}

value = transport->getValue(".1.3.6.1.2.1.1.1.0");
if (value == null)
{
    println("Failed to issue SNMP GET request");
    return 2;
}
else
{
    println("System description: " .. value);
    return 0;
}
```

Read SNMP octet string as byte

```
transport = $node->createSNMPTransport();
if (transport == null) exit;

varbind = transport->get(".1.3.6.1.2.1.25.3.5.1.2.1");
if (varbind == null) exit;

bytestream = varbind->getValueAsByteStream();

println(bytestream->pos); // after bytestream is created, it's position is set to 0
println("0x" .. d2x(bytestream->readByte(), 2)); // prints the hex value of 0-th byte
```

Read Table From Agent

This script can be put into Script Library and run from server's debug console. It accepts node object name or ID as first parameter, table name as second parameter, and prints content of given table to console.

```
// Find node object
node = FindObject($1);
if (node == null)
{
    println("ERROR: Node not found");
    return;
}

// REad table data from agent
table = AgentReadTable(node, $2);
if (table == null)
{
    println("ERROR: Cannot read table from agent");
    return;
}

// Print column names
for(i = 0; i < table->columnCount; i++)
    print("| " .. left(table->getColumnName(i), 20));
println("|");
for(i = 0; i < table->columnCount; i++)
    print("+" .. left("-", 21, "-"));
println("+");

// Print data
for(i = 0; i < table->rowCount; i++)
{
    for(j = 0; j < table->columnCount; j++)
    {
        print("| " .. left(table->get(i, j), 20));
    }
    println("|");
}
```

Recursively Collect Values from Custom Attributes

This script recursively collects values of custom attribute contacts from all node parents. Collected values concatenated into single string and separated by semicolons. Duplicate values added only once.

```
global contacts = "";  // concatenated values will be stored here
global presence = %{ };  // value presence indicator (hash map)
```

```
// walk through each parent object for current node
foreach(o : $node->parents)
{
    add_contacts(o);
}

// Concatenated result is in "contacts" global variable
println("Contacts: " .. contacts);

/**
 * Recursively add contacts from object and it's parents
 */
function add_contacts(curr)
{
    c = curr->getCustomAttribute("contacts");
    if ((c != null) && (presence[c] == null))
    {
        if (length(contacts) > 0)
            contacts = contacts .. ";" .. c;
        else
            contacts = c;
        presence[c] = true;
    }

    foreach(o : curr->parents)
    {
        add_contacts(o);
    }
}
```

Setting node geolocation from SNMP

Adjust the OIDs in SNMPGetValue as required.

```
transport = $node->createSNMPTransport();
if (transport == null) {
  return null;
}

lat = transport->get(".1.2.3.4.1");
lon = transport->get(".1.2.3.4.2");

if (lat == null || lon == null) {
  return null;
}

geoLoc = new GeoLocation(lat, lon);
$node->setGeoLocation(geoLoc);
```

```
return 0;
```

Object query to list asset values

Saved object query script to list all nodes linked to asset and list values for serial, vendor, model asset properties.

```
with
    objName (order = "asc", name = "Object name") = {
        return $node->name;
    },

    serial (name = "Serial") = {
        return $node->assetProperties->serial;
    },

    vendorVar (name = "Vendor") = {
        return $node->assetProperties->vendor;
    },

    model (name = "Model") = {
        return $node->assetProperties->model;
    }

(type == NODE) && $node->asset != null //show all nodes linked to asset
```

Exactly the same script, but without "with" syntax:

```
if (($object->type == NODE) && $node->asset != null)
{
    global objName (order = "asc", name = "Object name") = $node->name;
    global serial (name = "Serial") = $node->assetProperties->serial;
    global vendorVar (name = "Vendor") = $node->assetProperties->vendor;
    global model (name = "Model") = $node->assetProperties->model;
}
return ($object->type == NODE) && $node->asset != null; //show all nodes linked to
asset
```

Deprecated functions

This chapter contails deprecated functins that should not be used.

AgentExecuteCommand()

`AgentExecuteCommand(node, commandName, ⋯) ⇒ Boolean`

Execute agent command (action) on given node. Optional arguments starting from 3rd are passed as command arguments to the agent.

Prior to v. 4.2 this function was named AgentExecuteAction

Parameters

node	Node object instance (e.g. `$node`)
commandName	Name of the command to be executed
...	Optional arguments for command

Return

Boolean indicator of success

Example

```
>>> AgentExecuteCommand($node, "System.Restart");
true

>>> AgentExecuteCommand($node, "Custom.RestartService", "jetty9");
true

>>> AgentExecuteCommand($node, "nonexisting action");
false
```

AgentExecuteCommandWithOutput()

`AgentExecuteCommandWithOutput(node, commandName, ⋯) ⇒ String`

Execute agent command (action) on given node and collect standard output of the application defined by the command. Optional arguments starting from 3rd are passed as command arguments to the agent.

Prior to v. 4.2. this function was named AgentExecuteActionWithOutput.

Parameters

node	Node object instance (e.g. `$node`)
commandName	Name of the command to be executed

...	Optional arguments for command

Return

Output of the command or `null` if execution failed.

Example

```
>>> AgentExecuteCommandWithOutput($node, "Custom.Ping", "10.10.8.16");
PING 10.10.8.16 (10.10.8.16): 56 data bytes
64 bytes from 10.10.8.16: icmp_seq=0 ttl=64 time=0.084 ms
64 bytes from 10.10.8.16: icmp_seq=1 ttl=64 time=0.120 ms
64 bytes from 10.10.8.16: icmp_seq=2 ttl=64 time=0.121 ms
```

AgentReadList()

This function is deprecated starting from version 3.0. Please use method readAgentList of class Node.

`AgentReadList(node, name) ⇒ Array`

Request list metric directly from agent on given node.

Parameters

node	Node object instance (e.g. `$node`)
name	List name

Return

Array of strings or `null` if failed.

Example

```
>>> supportedLists = AgentReadList($node, "Agent.SupportedLists");
>>> foreach (l : supportedLists) { println(l); }
Agent.ActionList
Agent.SubAgentList
Agent.SupportedLists
Agent.SupportedParameters
Agent.SupportedPushParameters
...
```

AgentReadParameter()

This function is deprecated starting from version 3.0. Please use method readAgentParameter of class Node.

`AgentReadParameter(node, name) ⇒ String`

Request metric directly from agent on given node.

Parameters

node	Node object instance (e.g. `$node`)
name	Metric name

Return

String value or `null` if failed.

Example

```
>>> v = AgentReadParameter($node, "Agent.Version")
>>> println(v)
2.2.13
```

AgentReadTable()

This function is deprecated starting from version 3.0. Please use method readAgentTable() of class Node.

`AgentReadTable(node, name) ⇒ Table`

Request table metric directly from agent on given node.

Parameters

node	Node object instance (e.g. `$node`)
name	List name

Return

Instance of Table or `null` if failed.

Example

```
>>> t = AgentReadTable($node, "Agent.SubAgents");
>>> for (c : t->columns) {
>>>   print(c->name . " | ");
>>> }
>>> println("");
>>> for (row : t->rows) {
>>>   for(cell : row->values) {
>>>      print(cell . " | ");
>>>   }
>>>   println("");
>>> }
NAME | VERSION | FILE |
```

```
Darwin | 2.2.13 | darwin.nsm |
FILEMGR | 2.2.13-3-g4c02b65c50 | filemgr.nsm |
PING | 2.2.13-3-g4c02b65c50 | ping.nsm |
```

BindObject()

This function is deprecated starting from version 3.0. Please use NetObj::bind() or NetObj::bindTo() instead.

This function is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

```
BindObject(parent, child) => void
```

Bind all NetXMS objects that can be bound from console (nodes, subnets, clusters, and another containers) to container objects.

Parameters

parent	Parent object (NetObj referring to container object or infrastructure service root).
child	The NetXMS object to be linked to given parent object (Node or NetObj referring to subnet, container, or cluster).

Return

None.

Example

```
BindObject(FindObject(2), $node);    // Link current node directly to "Infrastructure
Services"
BindObject(FindObject("Services"), FindObject("Service_1"));    // Link object named
"Service_1" to container "Services"
```

CreateSNMPTransport()

This function is deprecated starting from version 3.0. Please use method createSNMPTransport of class Node.

```
CreateSNMPTransport(node, port, context, community) => SNMP_Transport
```

Create SNMP transport with communication settings defined on the node. It is possible to specify a community string but only community strings listed in Network Credentials will be accepted.

Parameters

node	Target node.
port	Optional parameter with port.
context	Optional parameter with context as a string.
community	Optional parameter with community string as a string.

Return

Instance of SNMPTransport or `null` if failed.

Example

```
>>> transport = CreateSNMPTransport(FindObject("Server1"))
>>> print transport->snmpVersion
2c
```

DeleteCustomAttribute()

This function is deprecated starting from version 3.0. Please use NetObj::deleteCustomAttribute() instead.

```
DeleteCustomAttribute(object, name) => void
```

Delete custom attribute `name` from `object`.

Parameters

object	Target object.
name	Name of the custom attribute.

Example

```
>>> DeleteCustomAttribute($node, "test")
>>> test@$node
null
```

DeleteObject()

This function is deprecated starting from version 3.0. Please use

NetObj::delete() instead.

This function is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

```
DeleteObject(object) => void
```

Delete object of class NetObj, Interface, or Node from the system.

Parameters

object	NetXMS object to be deleted. Can be instance of NetObj or any inherited (e.g. Node). Reference to the object can be obtained using FindObject() function.

Return

None.

Example

```
DeleteObject(FindObject("Service_1")); //delete object named Service_1
```

DriverReadParameter()

Deprecated since 3.0, use Node::readDriverParameter() instead.

```
DriverReadParameter(object, name) => String
```

Request driver-specific metric directly from network device driver (e.g. Rital). Works similarly to AgentReadParameter(), but query device driver insterad.

Parameters

object	node object
name	Name of the metric to query

Return

String value of the metric or `null` if request failed / metric not supported by driver

Example

```

```

EnterMaintenance()

This function is deprecated starting from version 3.0. Please use NetObj::enterMaintenance() instead.

```
EnterMaintenance(object) => void
```

Make an object enter Maintenance mode.

Parameters

object	NetObj that will be entered in maintenance mode

Return

None.

Example

```
EnterMaintenance($node);    // Enter current node in maintenance mode
EnterMaintenance(FindObject("Services"));    // Enter container "Services" in
maintenance mode
```

EventCodeFromName()

This function is deprecated starting from version 5.0.

```
EventCodeFromName(name) => Integer
```

Get event code from event name

Parameters

name	String	Event name

Return

Event code

Example

```
println(EventCodeFromName("SYS_NODE_DOWN")); // will print "28"
```

EventNameFromCode()

This function is deprecated starting from version 5.0.

```
EventNameFromCode(code) => String
```

Get event name from code

Parameters

code	Integer	Event code

Return

Event name

Example

```
println(EventNameFromCode(28)); // will print "SYS_NODE_DOWN"
```

ExpandString()

Use of this function is not recommended. Please use expandString method of NetObj, Event or Alarm instead.

```
ExpandString(string, object, event) => String
```

Expand string by replacing macros with their values.

Parameters

string	String to expand
object	Object. Optional, required to expand object-related macros
event	Event object. Optional, required to expand event-related macors

Return

Formated string

Example

```
>>> ExpandString("%v")
3.5.90
>>> ExpandString("%n", $node)
```

```
My node name
>>> ExpandString("%N", $node, $event)
SYS_THRESHOLD_REACHED
```

GetCustomAttribute()

This function is deprecated starting from version 3.0. Please use NetObj::getCustomAttribute() instead.

```
GetCustomAttribute(object, name) => String
```

Lookup custom attribute of the object.

Alternatively, attributes can be accessed as instance attribues (with `->`, attribute should exist) or by using `attribute@object` notion (which will return `null` instead of runtime error if attribue is missing).

Parameters

object	Object to query.
name	Name of the custom attribute.

Return

String value of the custom attribute of `null` if not found.

Example

```
>>> GetCustomAttribute($node, "test")
testvalue
>>> $node->test
testvalue
>>> test@$node
testvalue
```

GetInterfaceName()

This function is deprecated starting from version 3.4. Please use Node::getInterface(), Node::getInterfaceByIndex(), Node::getInterfaceByMACAddress() or Node::getInterfaceByName() to get interface object and name attribute to get it's name.

```
GetInterfaceName(node, interfaceIndex) => String
```

Get interface name by index

Parameters

node	Node	node object
interfaceIndex	Integer	interface index

Return

Interface name as a string

Example

```
println(GetInterfaceName($node, 1)); //Will print "lo"
```

GetInterfaceObject()

This function is deprecated starting from version 3.4. Please use Node::getInterface(), Node::getInterfaceByIndex(), Node::getInterfaceByMACAddress() or Node::getInterfaceByName() instead.

```
GetInterfaceObject(node, interfaceIndex) => Interface
```

Get interface object by index

Parameters

node	Node	node object
interfaceIndex	Integer	interface index

Return

Get node's interface by it's index

Example

```
interface = GetInterfaceObject($node, 1); //Will return interface with index 1
println(interface->ifIndex); //Will print "1"
```

GetNodeInterfaces()

This function is deprecated starting from version 3.0. Please use

interfaces attribute in Node.

```
GetNodeInterfaces(node) => void
```

Get all interfaces for given node.

Parameters

node	Object of class Node.

Return

Array of objects of class Interface, with first object placed at index 0.

Example

```
// Log names and ids of all interface objects for given node
interfaces = GetNodeInterfaces($node);
for(i : interfaces)
{
    trace(1, "Interface: name='" . i->name . "' id=" . i->id);
}
```

GetNodeParents()

This function is deprecated starting from version 3.0. Please use interfaces parents in NetObj.

```
GetNodeParents(node) => Array
```

Get accessible parent objects for given node.

Parameters

node	Node	Node object

Return

Array of objects of class NetObj or inherited from it, with first object placed at index 0. End of list indicated by array element with null value. Return value also affected by trusted nodes settings.

Example

```
// Log names and ids of all accessible parents for current node
parents = GetNodeParents($node);
foreach(p : parents)
{
```

```
    trace(1, "Parent object: name='" . p->name . "' id=" . p->id);
}
```

GetNodeTemplates()

This function is deprecated starting from version 3.0. Please use templates attribute of DataCollectionTarget.

```
GetNodeTemplates()
```

Get template objects applied on given node.

Parameters

node	Node object.

Return

Array of objects, with first object placed at index 0. Return value also affected by trusted nodes settings.

GetObjectChildren()

This function is deprecated starting from version 3.0. Please use children attribute in NetObj.

```
GetObjectChildren(object) => Array
```

Return array of child objects for the object.

Parameters

object	Target object.

Return

Array of NetObj instances.

Example

```
// Log names and ids of all accessible child objects for current node
children = GetObjectChildren($node);
for(p : children)
{
    trace(1, "Child object: name='" . p->name . "' id=" . p->id);
```

```
}
```

GetObjectParents()

This function is deprecated starting from version 3.0. Please use parents attribute in NetObj.

```
GetObjectParents(object) => Array
```

Return array of object parents.

Parameters

object	Target object.

Return

Array of NetObj instances.

Example

```
// Log names and ids of all accessible parents for current node
parents = GetObjectParents($node);
for(p : parents)
{
    trace(1, "Parent object: name='" . p->name . "' id=" . p->id);
}
```

index()

This function is deprecated starting from version 5.0. Please use String::indexOf().

```
index(string, substring, position) => Integer
```

Returns the position of the first occurrence of substring in string at or after position if specified. All index values are 1-based (i.e. the first character has index 1, not 0).

Parameters

string	String	The string which will be examined.
substring	String	The string which we will search for.

position	Integer	The starting position in the string to begin our search from the left. Optional parameter

Return

Integer value of the position substring was found at, will return 0 if not found.

Example

```
println(index("abcdef","cd")); //Will print "3"
println(index("abcdef","cd",4)); //Will print "0"
println(index("abcdefabcdef","cd",4)); //Will print "9"
```

Instance()

This function is deprecated starting from version 5.0. Please return array wit instance values.

```
Instance(name, displayName, object) => Array
```

This is helper function for instance filter script. It can be used to return accepted item. This function has named parameters.

Parameters

name	String	Instance name ({instance})
displayName	String	Instance display name ({instance-name})
object	NetObj	Related object

Return

Array, where the first value is `true`, the second is `name`, the third is `displayName` and the forth is `object`.

Example

```
return Instance(displayName: GetInterfaceName($node, $1), object:
GetInterfaceObject($node, $1), name: $1); //This will return correctly formatted array
to accept instance
```

LeaveMaintenance()

This function is deprecated starting from version 3.0. Please use NetObj::leaveMaintenance() instead.

```
LeaveMaintenance(object) => void
```

Make an object leave Maintenance mode.

Parameters

object	NetObj that will leave maintenance mode

Return

None.

Example

```
LeaveMaintenance($node);    // Make current node leave maintenance mode
LeaveMaintenance(FindObject("Services"));    // Make container "Services" leave
maintenance mode
```

left()

This function is deprecated starting from version 5.0. Please use String::left().

```
left(string, length, pad) => String
```

Returns the string of length characters of string, optionally padded with pad character instead of a blank (space).

Parameters

string	String	The string which will be processed.
length	Integer	The number of character to return, must be a positive integer.
pad	String	The pad character to use instead of blank spaces. Optional parameter.

Return

String of the left length characters.

Example

```
println(left("abc d",8)); //Will print "abc d   "
println(left("abc d",8,".")); //Will print "abc d..."
println(left("abc  def",7)); //Will print "abc  de"
```

length()

This function is deprecated starting from version 5.0. Please use String::length.

```
length(string) => Integer
```

Returns the length of string.

Parameters

string	String	The string to determine its length.

Return

Length of the string passed to the function.

Example

```
println(length("abcd")); // Will print "4"
```

lower()

This function is deprecated starting from version 5.0. Please use String::toLowerCase().

```
lower(string) => String
```

Converts string to lowercase string.

Parameters

string	String	String to convert

Return

Source string converted to lowercase.

Example

```
println(lower("aBcD")); // Will print "abcd"
```

ltrim()

This function is deprecated starting from version 5.0. Please use String::trimLeft().

```
ltrim(string) => String
```

Removes blanks (space and tab characters) from the left side of specified string.

Parameters

string	String	String to trim

Return

Source string with blanks at the left side removed.

Example

```
ltrim("  abc def  "); // Will print "abc def  "
```

ManageObject()

This function is deprecated starting from version 3.0. Please use manage functions in NetObj.

```
ManageObject(object) => void
```

Set object into managed state. Has no effect if object is already in managed state.

Parameters

object	NetXMS object to be modified. Can be NXSL class NetObj or any inherited for it. Reference to object can be obtained using FindObject() function.

Example

```
ManageObject(FindObject(125));    // Set object with id 125 to managed state
```

RenameObject()

This function is deprecated starting from version 3.0. Please use rename

method in NetObj.

```
RenameObject(object, name) => void
```

Rename object.

Parameters

object	NetXMS object to be renamed. Can be NXSL class NetObj or any inherited for it. Reference to object can be obtained using FindObject function.
name	New name for object.

Return

None.

Example

```
RenameObject(FindObject(2), "My Services");    // Rename "Infrastructure Services"
object
```

replace()

This function is deprecated starting from version 5.0. Please use String::replace().

```
replace(string, old, new) => String
```

Returns the string with all the occurrences of old substring replaced with the new substring. The replacement proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaa" will result in "ba" rather than "ab".

Parameters

string	String	The string which will be processed.
old	String	Old substring to be replaced
new	String	The new substring, which would replace the old substring. Can be an empty string.

Return

Source string with substrings replaced.

Example

```
println(replace("one four three","four","two")); //Will print "one two three"
println(replace("abc  def"," ",""")); //Will print "abcdef"
```

right()

This function is deprecated starting from version 5.0. Please use String::right().

```
right(string, length, pad) => String
```

Returns the string of length characters of string, optionally padded with pad character instead of blank (space) starting from the right. Padding occurs on the left portion of the string.

Parameters

string	String	The string which will be processed.
length	Integer	The number of character to return, must be a positive integer.
pad	String	The pad character to use instead of blank spaces. Option parameter.

Return

String of the right length characters.

Example

```
println(right("abc  d",8)); //Will print "  abc  d"
println(right("abc def",5)); //Will print "c def"
println(right("17",5,"0")); //Will print "00017"
```

rindex()

This function is deprecated starting from version 5.0. Please use String::lastIndexOf().

```
rindex(string, substring, position) => Integer
```

Returns the position of the last occurrence of substring in string up to or before position if specified. All index values are 1-based (i.e. the first character has index 1, not 0).

Parameters

string	String	The string which will be examined.
substring	String	The string which we will search for.
position	Integer	The position in string to start searching back from. Optional parameter.

Return

Integer value of the position substring was found at, will return 0 if not found.

Example

```
println(rindex("abcdabcd","cd")); // Will print "7"
println(rindex("abcdef","cd",2)); // Will print "0"
println(rindex("abcdefabcdef","cd",4)); // Will print "3"
```

rtrim()

This function is deprecated starting from version 5.0. Please use String::trimRight().

```
rtrim(string) => String
```

Removes blanks (space and tab characters) from the right side of specified string.

Parameters

string	String	Source string

Return

Source string with blanks at the right side removed.

Example

```
println(rtrim("  abc def  ")); //Will print "  abc def"
```

SetCustomAttribute()

This function is deprecated starting from version 3.0. Please use setCustomAttribute method in NetObj.

```
SetCustomAttribute(object, name, value) => void
```

Set custom attribute `name` to `value` on `object`.

Parameters

object	Target object.
name	Custom attribute name.
value	Custom attribute value.

Example

```
>>> SetCustomAttribute($node, "test", "test value")
>>> test@$node
test value
```

SetEventParameter()

This function is deprecated starting from version 5.0. Please use setNamedParameter method in Event.

```
SetEventParameter(event, parameterName, value) => void
```

Set value of event's named parameter.

Parameters

event	Event	Event object, you can use predefined variable $event to refer to current event.
parameterName	String	Parameter's name.
value	String	New value.

Return

No return value

Example

```
SetEventParameter($event, "customParameter", "new value");
```

SetInterfaceExpectedState()

This function is deprecated starting from version 3.0. Please use setExpectedState functions in Interface.

```
SetInterfaceExpectedState() => void
```

Set expected state for given interface.

Parameters

interface	Interface object. Can be obtained using GetNodeInterfaces or GetInterfaceObject.
state	New expected state for interface. Can be specified as integer code or state name. Interface expected states

Return

None.

Example

```
// Set expected state to "ignore" for all interfaces of given node
interfaces = GetNodeInterfaces($node);
foreach(i : interfaces)
{
    SetInterfaceExpectedState(i, "IGNORE");
}
```

SNMPGet()

This function is deprecated starting from version 3.0. Please use method get of class SNMPTransport.

```
SNMPGet(transport, oid) => SNMP_VarBind
```

Perform SNMP GET request for `oid` over provided transport.

Parameters

transport	Transport created by CreateSNMPTransport().
oid	SNMP OID string.

Return

Instance of SNMPVarBind or `null` on failure.

Example

```
>>> transport = CreateSNMPTransport(FindObject("Server1"));
>>> if (transport != null) {
>>>   oid = ".1.3.6.1.2.1.25.1.6.0"; // number of running processes
>>>   varbind = SNMPGet(transport, oid);
>>>   if (varbind != null) {
>>>     trace(1, varbind->name . "=" . varbind->value);
>>>   }
>>>   else {
>>>     trace(0, "SNMPGet() failed");
>>>   }
>>> }
```

SNMPGetValue()

This function is deprecated starting from version 3.0. Please use method getValue of class SNMPTransport.

```
SNMPGetValue(transport, oid) => String
```

Perform SNMP GET request for `oid` over provided transport and return single string value instead of varbind.

This function is a wrapper for SNMPGet().

Parameters

transport	Transport created by CreateSNMPTransport().
oid	SNMP OID string.

Return

String value of the result or `null` on failure.

Example

```
>>> transport = CreateSNMPTransport(FindObject("Server1"));
>>> if (transport != null) {
>>> oid = ".1.3.6.1.2.1.25.1.6.0"; // number of running processes
>>> value = SNMPGetValue(transport, oid);
>>> if (value != null) {
>>>     trace(1, value);
>>> }
>>> else {
>>>     trace(0, "SNMPGetValue() failed");
>>> }
```

```
>>> }
```

SNMPSet()

This function is deprecated starting from version 3.0. Please use method set of class SNMPTransport.

```
SNMPSet(transport, oid, value, dataType) => Boolean
```

Perform SNMP SET request for `oid` over provided transport. Return boolean success indicator. `value` is automatically converted from string based in `dataType`. If `dataType` is not provided, default type "STRING" will be used.

Parameters

transport	Transport created by CreateSNMPTransport().
oid	SNMP OID string.
value	New value.
dataType	Type of the `value`, default to "STRING". <<SNMP data types

Return

Boolean. TRUE on success and FALSE in case of failure.

Example

```
>>> if (!SNMPSet(transport, oid, "192.168.0.1", "IPADDR") {
>>>     trace(1, "SNMPSet failed");
>>> }
```

SNMPWalk()

This function is deprecated starting from version 3.0. Please use method set of class SNMPTransport.

```
SNMPWalk(transport, oid) => Array
```

Perform SNMP WALK request for `oid` over provided transport and return collected values as array of SNMPVarBind or `null` on failure.

Parameters

transport	Transport created by CreateSNMPTransport().
oid	SNMP OID string.

Return

Array of SNMPVarBind or `null` or failure.

Example

```
>>> transport = CreateSNMPTransport(FindObject("Server1"));
>>> if (transport != null) {
>>>     oid = ".1.3.6.1.2.1.25.4.2.1.2"; // Names of the running processes
>>>     vars = SNMPWalk(transport, oid);
>>>     if (vars != null) {
>>>         foreach (v: vars) {
>>>             trace(1, v->name."=".v->value);
>>>         }
>>>     }
>>> }
```

SplitString()

This function is deprecated starting from version 5.0. Please use String::split().

```
SplitString(string, separator) => Array
```

Split string into array of strings at given separator.

Parameters

string	String	String to split.
separator	String	Separator character. If supplied string is longer than 1 character, it's first character will be used as separator.

Return

Array with strings

Example

```
SplitString("a;b;c;d", ";"); //Will be splited to %("a", "b", "c", "d")
SplitString("abcd", ";"); //Will be splited to %("abcd")
```

substr()

This function is deprecated starting from version 5.0. Please use String::substring().

```
substr(string, n, len) => void
```

Extracts the substring from string that begins at the nth character and is of length len.

Parameters

string	String	Source string.
n	Integer	Starting character index for substring. The n must be a positive whole number. If n is greater than length(string), then empty string is returned.
len	Integer	Length of substring. If you omit length, the rest of the string is returned. Optional parameter.

Return

Extracted substring.

Example

```
print(substr("abcdef", 3, 2)); //Will print: "cd"
print(substr("abcdef", 8)); //Will print: ""
print(substr("abcdef", 4)); //Will print: "def"
```

trim()

This function is deprecated starting from version 5.0. Please use String::trim().

```
trim(string) => String
```

Removes blanks (space and tab characters) from both sides of specified string.

Parameters

string	String	String to trim

Return

Source string with blanks at both sides removed.

Example

```
print(trim("  abc def  ")); //will print "abc def"
```

UnbindObject()

This function is deprecated starting from version 3.0. Please use NetObj::unbind() and NetObj::unbindFrom() instead.

This function is enabled by default, but can be disabled by setting configuration parameter "NXSL.EnableContainerFunctions".

```
UnbindObject(parent, child) => void
```

Remove (unbind) object from a container.

Parameters

parent	Parent object (NetObj referring to container object or infrastructure service root).
child	The NetXMS object to be unlinked from given parent object (Node or NetObj referring to node, subnet, container, or cluster).

Return

None.

Example

```
UnbindObject(FindObject("Services"), FindObject("Service_1"));    // Unlink object
named "Service_1" from container "Services"
```

UnmanageObject()

This function is deprecated starting from version 3.0. Please use unmanage functions in NetObj.

```
UnmanageObject(object) => void
```

Set object into unmanaged state. Has no effect if object is already in unmanaged state.

Parameters

object	NetXMS object to be modified. Can be NXSL class NetObj, Node, or Interface. Reference to object can be obtained using FindObject function.

Return

None.

Example

```
UnmanageObject(FindObject(2));    // Set "Infrastructure Services" object to unmanaged
state
```

upper()

This function is deprecated starting from version 3.0. Please function String::toUpperCase().

```
upper(string) => String
```

Converts string to uppercase.

Parameters

string	String	String to convert

Return

Source string converted to uppercase.

Example

```
print(upper("aBcD")); //will print "ABCD"
```