NConf perl-API

NAME

The NConf perl-module (and sub-modules) are a collection of shared functions to be used in perl scripts surrounding the NConf software. They offer an API to NConf internal functionality, namely the database.

SYNOPSIS

```
use NConf;
setLoglevel($loglevel);

use NConf::Logger;
logger($loglevel, $msg);

use NConf::ImportNagios;
%data_ref = parseNagiosConfigFile($class_name, $filename);

use NConf::ImportCsv;
%data_ref = parseCsv($filename, $class_name, $delimiter);
%data_ref = parseHostServiceCsv($filename, $delimiter);

use NConf::Helpers;

$scalar = readNConfConfig($filename, $php_const_name, 'scalar');
@array  = readNConfConfig($filename, $php_array_name, 'array');
@dist_array  = makeValuesDistinct(@ref_array);
$str_out = replaceMacros($str_in);

use NConf::DB;

setDbReadonly(1);
$dbh = dbConnect();
dbDisconnect();
$q_str = dbQuote($str);

use NConf::DB::Read;

$id_item     = getItemId($item_name, $item_class);
$id_srv      = getServiceId($srv_name, $id_host);
$host_name   = getServiceHostname($id_service);
$id_attr     = getAttrId($attr_name, $attr_class);
$naming_attr = getNamingAttr($class_name);

$item_name   = getItemName($id_item);
$item_class  = getItemClass($id_item);
@ref_array   = getItemData($id_item);
@ref_array   = getItems($class_name);
@ref_array   = getItemsLinked($id_item);
@ref_array   = getChildItemsLinked($id_item);

%conf_attrs  = getConfigAttrs();
%conf_class  = getConfigClasses();
$unique_name = getUniqueNameCounter($class_name, $item_name);

$retval      = checkItemsLinked($id_item, $id_linked, $attr_name);
$retval      = checkLinkAsChild($id_attr);
$retval      = checkItemExistsOnServer($id_item2check, $id_collector);
```

```
$scalar      = queryExecRead($sql, $msg, 'one');
@array       = queryExecRead($sql, $msg, 'row');
%hash        = queryExecRead($sql, $msg, 'row2');
@ref_array   = queryExecRead($sql, $msg, 'all');
%ref_hash    = queryExecRead($sql, $msg, 'all2', $key_field);

use NConf::DB::Modify;

$retval = addItem($item_class, %data);
$retval = linkItems($id_item, $id_linked, $attr_name);
$retval = insertValue($item_class, $id_item, $attr_name, $attr_value);

$retval = addHistory($action, $attr_name, $attr_value, $id_item);
$retval = addHistory($action, $class_name, $item_name, $id_item);

 $qres = queryExecModify($sql, $msg);
($qres, $id_item) = queryExecModify($sql, $msg, 1, $class_name);
```

DESCRIPTION

The NConf perl-API offers the following functionality:

NConf

The main perl-module.

Parameters:

$NC_loglevel (default value: 3 → 'INFO')

Defines the global loglevel for all messages displayed by the logger. This parameter should always be set using the setLoglevel() function.

NConf::Logger

Offers a logger() function which logs and displays messages based on the global loglevel parameter.

NConf::ImportNagios

A preliminary parser for Nagios configuration files.

NConf::ImportCsv

A parser for CSV files.

NConf::Helpers

Miscellaneous functions, such as a parser for the NConf configuration.

NConf::DB

The main database API. Provides basic database functionality.

Parameters:

$NC_db_readonly (default value: 0)

Defines if write operations should be executed in the database or not. This is useful if you want to provide a "simulation"

mode in your scripts. This parameter should always be set using the setDbReadonly() function.

$NC_db_caching (default value: 1)

Defines if the API should cache certain values it reads from the database. The API is intelligent enough to only cache values that are not likely to change in the course of script execution, such as attribute and datatype information. This parameter is set statically in the NConf::DB module.

NConf::DB::Read

Provides functions which execute read-only queries in the database.

NConf::DB::Modify

Provides functions which actually modify data in the database. Also provides functions for maintaining the NConf history.

FUNCTIONS

NConf

setLoglevel()

```
setLoglevel($loglevel);
```

Function use: Set the overall loglevel

Expected arguments:

1. The loglevel to set (1-5)

NConf::Logger

logger()

```
logger($loglevel, $msg);
```

Function use: Log and display messages based on loglevel

Expected arguments:

1. The loglevel of the message (1-5)
2. The message to be logged
3. Optional: the type of message ('1'=sql)

CAUTION: loglevel 1 is percieved to be "fatal". Execution will be terminated!

The loglevels are defined as follows:

1. [ERROR] fatal errors which terminate script execution
2. [WARN] warnings which require the user's attention
3. [INFO] information such as progress and success messages
4. [DEBUG] step by step information of what's going on
5. [TRACE] detailed step by step information with function calls and complete SQL queries

---

NConf::ImportNagios

parseNagiosConfigFile()

```
%data_ref = parseNagiosConfigFile($class_name, $filename);
```

Function use: parse a Nagios config file, load data into memory

Expected arguments:

1. NConf class name of items to parse for
2. File to parse

Return values:

1. Returns a hash containing all naming-attr values as the key, and a reference to a hash containing any attr→value pairs as the value

---

NConf::ImportCsv

parseCsv()

```
%data_ref = parseCsv($filename, $class_name, $delimiter);
```

Function use: parse a CSV file, load data into memory

Expected arguments:

1. CSV file to parse
2. the NConf class name of items to parse for
3. optional: the CSV delimiter (defaults to ”;”)

Return values:

1. Returns a hash containing all naming-attr values as the key, and a reference to a hash containing any attr→value pairs as the value

---

parseHostServiceCsv()

```
%data_ref = parseHostServiceCsv($filename, $delimiter);
```

Function use: parse a CSV file containing host/service information, load data into memory

The CSV file must have the following format:

```
host_name;attr 1;attr 2;service_description;attr 1;attr 2[;service_description;...;...]
<---------------------><------------------------------><--------------------------->
     host attributes            service attributes             additional services
```

The attribute names must be specified within the header line of the CSV file. You may specify any number of host and service attributes you'd like to import (consider mandatory attributes). The import process will assume that any attributes which follow the “service_description” attribute belong to that service.

Expected arguments:

1. CSV file to parse
2. optional: the CSV delimiter (defaults to ”;”)

Return values:

1. Returns a hash containing all naming-attr values as the key, and a reference to a hash containing any attr→value pairs as the value

---

NConf::Helpers

readNConfConfig()

```
$scalar = readNConfConfig($filename, $php_const_name, 'scalar');
@array  = readNConfConfig($filename, $php_array_name, 'array');
```

Function use: fetch configuration options from the main NConf configuration

Expected arguments:

1. configuration file name and path
2. configuration option (PHP constant) name
3. type of value to read (scalar|array)
4. optional: '1' = WARN instead of exiting with ERROR if option was not found in config

Return values:

1. either a scalar or an array containing the values read from the config

---

makeValuesDistinct()

```
@dist_array  = makeValuesDistinct(@ref_array);
```

Function use: make sure that identical attribute values are combined to a single comma-separated string

Expected arguments:

1. an array containing references to arrays
   that contain 'attr'→'value' pairs

```
Example:
arrayref -> array([0] "attr_X", [1] "value1")
arrayref -> array([0] "attr_X", [1] "value2")
arrayref -> array([0] "attr_Y", [1] "value3")
arrayref -> array([0] "attr_Y", [1] "value4")
```

Return values:

1. an array containing references to arrays holding
   a comma separated list of values for each attr

```
Example:
arrayref -> array([0] "attr_X", [1] "value1,value2")
arrayref -> array([0] "attr_Y", [1] "value3,value4")
```

---

replaceMacros()

```
$str_out = replaceMacros($str_in);
```

Function use: replace any %..% style NConf macros in a string with their respective value(s)

Expected arguments:

1. the string to search for macros

Return values:

1. the string with the replaced macros

This function looks for %…% style macros in any string that was passed to it. If it finds a macro, it will look for a replacement value in the globally defined hash '%NC_macro_values'. The hash is expected to have been defined in advance by the component calling the replaceMacros() function.

The hash structure is the following:
$NC_macro_values{'MACRO_NAME'} → 'replacement value'

NConf::DB

setDbReadonly()

```
setDbReadonly(1);
```

Function use: prevent any actual modifications from being made to the database

Expected arguments:

1. 1 = set to read-only, 0 = unset

---

dbConnect()

```
$dbh = dbConnect();
```

Function use: connect to NConf database, if not yet connected

No arguments expected.

Return values:

1. returns a DBI:mysql database handle

---

dbDisconnect()

```
dbDisconnect();
```

Function use: disconnect from NConf database, if connected

No arguments expected.

---

dbQuote()

```
$q_str = dbQuote($str);
```

Function use: quote and excape any special chars for safe SQL usage

Expected arguments:

1. string to be quoted

Return values:

1. quoted string

NConf::DB::Read

getItemId()

```
$id_item = getItemId($item_name, $item_class);
```

Function use: fetch the ID of any item in the database

Expected arguments:

1. the item name (the contents of the naming-attr)
2. the class of the item

Return values:

1. a scalar containing the ID of the item, undef on falure

---

getServiceId()

```
$id_srv = getServiceId($srv_name, $id_host);
```

Function use: fetch the ID of a service in the database

Expected arguments:

1. the service name (the contents of the naming-attr)
2. the ID of the parent host (value in 'host_name' attr)

Return values:

1. a scalar containing the ID of the service, undef on falure

---

getServiceHostname()

```
$host_name = getServiceHostname($id_service);
```

Function use: fetch the hostname of the host which the service belongs to

Expected arguments:

1. the ID of the service

Return values:

1. the hostname of the host which the service belongs to

Info: a service always belongs to exactly one host (1:1)

---

getAttrId()

```
$id_attr = getAttrId($attr_name, $attr_class);
```

Function use: fetch attr ID based on attr name and class name

Expected arguments:

1. attribute name
2. class name

Return values:

1. attr ID, undef on failure

---

getNamingAttr()

```
$naming_attr = getNamingAttr($class_name);
```

Function use: get the name of the "naming attribute" for a specific class

Expected arguments:

1. class name

Return values:

1. attribute name of the "naming attribute" for the specified class (e.g. 'host_name' for class 'host')

---

getItemName()

```
$item_name = getItemName($id_item);
```

Function use: fetch the name (the value of the naming attr) of any item in the database

Expected arguments:

1. the item ID

Return values:

1. a scalar containing the name of the item, undef on falure

---

getItemClass()

```
$item_class = getItemClass($id_item);
```

Function use: fetch the class of an item based on the item ID

Expected arguments:

1. item ID

Return values:

1. class name, undef on failure

---

getItemData()

```
@ref_array  = getItemData($id_item);
```

Function use: fetch all attributes and values assigned to an item over the ConfigValues table
(i.e. the data of 'text', 'select' and 'password' attributes)

Expected arguments:

1. the item ID

Return values:

1. an array containing references to arrays that contain 'attr' → 'value' pairs

---

getItems()

```
@ref_array  = getItems($class_name);
```

Function use: fetch all items of a certain class (e.g. all contacts)

Expected arguments:

1. class name
2. optional: '1' = also return item names

Return values:

1. an array containing references to arrays with two values:
   [0] the item ID
   [1] optional: the name of the item (value of the naming attr)

---

getItemsLinked()

```
@ref_array  = getItemsLinked($id_item);
```

Function use: fetch all attributes and values linked to an item over the ItemLinks table (i.e. the data of 'assign_one', 'assign_many' and 'assign_cust_order' attributes)

Expected arguments:

1. the item ID

Return values:

1. an array containing references to arrays that contain the following data structure:
   [0] the attribute name
   [1] the attribute value (the name of the linked item)
   [2] the 'write_to_conf' flag for the current attribute
   [3] the item ID of the linked item
   [4] the order number of the linked item within an 'assign_cust_order' attribute
   [5] the ordering number of the current attribute within its class

CAUTION: Be careful when using getItemsLinked() in conjunction with makeValuesDistinct(). The latter function is very useful in grouping multiple items linked over the same attribute, but makeValuesDistinct() tends to mess up the rest of the output from getItemsLinked(), because it only works with the first two values (the attribute name and attribute value).

---

getChildItemsLinked()

```
@ref_array  = getChildItemsLinked($id_item);
```

Function use: fetch all child items linked to another item
(the items it is being used by, e.g fetch all items that use a certain timeperiod).

Expected arguments:

1. the item ID of the parent item

Return values:

1. an array containing references to arrays that contain the following data structure:
   [0] the item ID of the linked child item
   [1] the name of the child item
   [2] the class of the child item

---

getConfigAttrs()

```
%conf_attrs = getConfigAttrs();
```

Function use: get a list of all attrs plus their properties

No arguments expected.

Return values:

1. A hash containing the following data structure:
   $conf_attrs{'class name'}→{'attr name'}→{'property'}

The following properties exist:

- 'id_attr': the attribute ID
- 'friendly_name': the attribute name displayed in the GUI
- 'description': the attribute description displayed in the GUI
- 'datatype': the data type ('text', 'select', 'assign_one' etc.)
- 'max_length': the maximum amount of chars (for text attributes)
- 'poss_values': a list of possible values (for select attributes)
- 'predef_value': the predefined value(s)
- 'mandatory': if the attribute is mandatory or not
- 'ordering': the attribute's ordering number within its class
- 'visible': if the attribute is visible in the GUI
- 'write_to_conf': if the attribute should be written to the generated config
- 'naming_attr': if the attribute is a "naming attribute"
- 'link_as_child': if the 'link_as_child' flag is set
- 'link_bidirectional': if the 'link_bidirectional' flag is set
- 'fk_show_class_items': for linking attributes, which class to link to (class ID)
- 'assign_to_class': for linking attributes, which class to link to (class name)

---

getConfigClasses()

```
%conf_class = getConfigClasses();
```

Function use: get a list of all classes plus their properties

No arguments expected.

Return values:

1. A hash containing the following data structure:
   $class_hash{'class name'}→{'property'}

The following properties exist:

- 'class_type': whether the class is “global”, “monitor” or “collector” specific
- 'out_file': the name of the .cfg file to write items of this class to
- 'nagios_object': the object definition for Nagios ('define host {')

---

getUniqueNameCounter()

```
$unique_name = getUniqueNameCounter($class_name, $item_name);
```

Function use: determine a unique name for items by appending a numeric counter, if necessary

Expected arguments:

1. the class name
2. the current item name that needs to be made unique

Return values:

1. a unique name for a new item within that class in the following format:

“item name_n” (“n” being a unique numeral)

Info: this function will query the database the first time it processes a new item name (and for double-checking). It will store the last generated counter value in a global cache and will access the cache if it is invoked with the same parameters a second time. This allows counters to remain unique even if items aren't commited to the database immediately.

---

checkItemsLinked()

```
$retval = checkItemsLinked($id_item, $id_linked, $attr_name);
```

Function use: check if two items are linked via the ItemLinks table

Expected arguments:

1. ID of item that will be linked
2. ID of item to link the first one to
3. name of NConf attr (of type assign_one/many)

Return values:

1. - 'true' if items already linked,
   - 'false' if not,
   - undef on failure

This function automatically checks and considers the “link_as_child” flag

---

checkLinkAsChild()

```
$retval = checkLinkAsChild($id_attr);
```

Function use: check if “link_as_child” flag is set for a specific attribute

Expected arguments:

1. attr ID

Return values:

1.
   - 'true' if link_as_child = “yes”,
   - 'false' if link_as_child = “no”,
   - undef on failure

---

checkItemExistsOnServer()

```
$retval = checkItemExistsOnServer($id_item2check, $id_collector);
```

Function use: check if a server-specific item (host, service, hostgroup or servicegroup) exists on a collector server, or if it's monitored at all. This function is intended for distributed Nagios environments with multiple collectors.

Expected arguments:

1. ID of item (host or service) to check for
2. optional: ID of a specific collector server

If the second parameter is specified, the function will determine if the item being checked exists on the specified collector server.
If the second parameter is omitted, the function will determine if the item being checked is monitored at all by any collector (the “monitored by” flag is set for hosts / “service_enabled” is not set to “no” for services).

Return values:

1.
   - 'true' if item exists on specified collector / if item is monitored by any collector,
   - 'false' if item does not exist on collector / if item is not monitored by any other collector

Note: this function will always return 'true' if the item being checked is NOT a host, service, hostgroup or servicegroup.

---

queryExecRead()

```
$scalar     = queryExecRead($sql, $msg, 'one');
@array      = queryExecRead($sql, $msg, 'row');
%hash       = queryExecRead($sql, $msg, 'row2');
@ref_array  = queryExecRead($sql, $msg, 'all');
%ref_hash   = queryExecRead($sql, $msg, 'all2', $key_field);
```

Function use: Execute a query which reads data from the database

Expected arguments:

1. The SQL query
2. The message to log for the query
3. The format of the return value:
   - “one” return a single scalar value (first value returned by the query)
   - “row” return an array containing all values of one row
   - “row2” return a hash containing all values of one row (with attr names as keys)
   - “all” return an array that contains one array reference per row of data
   - “all2” return a hash containing one hash reference per row of data (with a specified attr as key)
4. Only if “all2”: The attr name to use as key for the hash that is returned

Return values:

1. The output of the query in the specified format, undef if query returns no rows / on failure

---

NConf::DB::Modify

addItem()

```
$retval = addItem($item_class, %data);
```

Function use: add a new item including all its attributes / values

Expected arguments:

1. The class name of the item you want to add (e.g. 'host, 'contact' etc.)
2. A hash containing all attr/value pairs of the item to be added

Return values:

1. 'true' on operation success, undef on failure

---

linkItems()

```
$retval = linkItems($id_item, $id_linked, $attr_name);
```

Function use: link two items in the ItemLinks table

Expected arguments:

1. ID of item that will be linked
2. ID of item to link the first one to
3. name of NConf attr (of type assign_one/many)
4. optional: order number for assign_cust_order attributes

Return values:

1. 'true' on operation success, undef on failure

This function automatically checks and considers the “link_as_child” flag. It also links items in the proper order, if datatype is “assign_cust_order”.

---

insertValue()

```
$retval = insertValue($item_class, $id_item, $attr_name, $attr_value);
```

Function use: add the value of a single attribute (any datatype)

Expected arguments:

1. class name
2. ID of item
3. attribute name
4. attribute value

Return values:

1. 'true' on operation success, undef on failure

---

addHistory()

```
$retval = addHistory($action, $attr_name, $attr_value, $id_item);
$retval = addHistory($action, $class_name, $item_name, $id_item);
```

Function use: add a history entry

Expected arguments:

1. action
2. attr name or class name
3. attr value or item name
4. item ID

Return values:

1. 'true' on operation success, undef on failure

---

queryExecModify()

```
$qres = queryExecModify($sql, $msg);
($qres, $id_item) = queryExecModify($sql, $msg, 1, $class_name);
```

Function use: execute a query which modifies data in the database

Expected arguments:

1. the SQL query
2. the message to log for the query
3. optional: '1' if the insert_id should be returned
4. if param 3 was passed, also pass the class name of the item being inserted (not optional)

Return values:

1. 'true' on operation success, undef on failure
2. the ID of the newly inserted record (if requested)

AUTHOR

Written by Angelo Gargiulo

COPYRIGHT

SEE ALSO

Project homepage: http://www.nconf.org [http://www.nconf.org]