

The NO-SOD Framework 2.0.2 Developer Manual

Stephane Van de Putte, The NO-SOD Project

The NO-SOD Framework 2.0.2 Developer Manual

by Stephane Van de Putte

Published 2004

Copyright © 1999, 2000, 2001, 2002, 2003, 2004 The NO-SOD Project

Table of Contents

1. Basic rules for module development	1
INTRODUCTION	1
MODULE CODE	1
DATABASE	1
File naming convention	1
Variables naming convention	2
SESSION VARIABLE - REGISTERED	2
RECEIPT DATA FROM HTML	2
OUTPUT GENERATION	2
SOURCE CODE CALLING	2
CERTIFICATE	3
2. FLY.PHP - RUNTIME EXECUTION WORKFLOW	4
STEP 1: Set up environment variables	4
STEP 2: Including external files	5
STEP 3: Get the client IP addresss	5
STEP 4: Check the current client browser window	5
STEP 5: Start a new session if necessary	5
STEP 6: Fo non new sessions: take back session flow	6
STEP 7: Source code extraction and execution	6
STEP 8: Output by merging html template.	7
STEP 9: Serializing objects	7
STEP 10: Output error if any.	7
3. COMMONLY USED VARIABLES	8
4. HTMLGEN - Generating an html table dynamically	9
5. HTMLGENTABLE - Generating an html table dynamically with row effect	10
6. The User Object definition - class nsdUser	12
Object properties	12
Object methods	13
get([\$load=1])	13
add([\$load=1],[\$reason="ownership"])	13
delete([\$load=1],[\$reason="ownership"])	13
undelete([\$load=1],[\$reason="ownership"])	13
.....	13
update([\$load=1],[\$passupdate=false],[\$unlock=false],[\$reason="ownership"])	14
.....	14
getuname(\$uid)	14
tsig(\$user,\$password,[\$userstorage=1])	14
login(\$user,\$password,[\$module="NSD"],[\$userstorage=1])	14
7. The Group Object definition - class nsdGroup	15
Linking users to groups.	15
Object properties	15
Object methods	15
add(\$gname,\$module,\$type,[\$gdesc=""],[\$reason="ownership"])	15
getgid(\$gname,\$module,\$type)	16
getgname(\$gid)	16
update(\$oldname,\$gname,\$module,\$type,[\$gdesc=""])	16
adduser(\$uname,\$gname,\$module,\$type,[\$reason="ownership"])	16
deluser(\$uname,\$gname,\$module,\$type,[\$reason="ownership"])	17
listusersofgroup(\$gname,\$module,\$type)	17
listgroupsofuser(\$uname,\$module,\$type)	17
listgroupsofmodule(\$module,\$type)	17
userbelong2group(\$uname,\$gname,\$module,\$type)	18
ad-	
dgroup(\$ngname,\$nmodule,\$ntype,\$gname,\$module,\$type,[\$reason="ownership"	
])	18
del-	
group(\$ngname,\$nmodule,\$ntype,\$gname,\$module,\$type,\$reason="ownership")	

.....	18
listsubgroupsofgroup(\$gname,\$module,\$type,\$nmodule,\$ntype)	19
listgroupsmanagedbyuser(\$uname,\$module)	19
listmastergidofuser(\$uname)	19
8. HOW LIBRARY (.LIB) FILES ARE WORKING ?	20

List of Tables

3.1. Variables list	8
---------------------------	---

Chapter 1. Basic rules for module development

INTRODUCTION

The following rules apply to the NO-SOD Framework version 2.0.1 (tod10), and are presented as a summary of a starting point for new developers. This document is not the complete developer manual, but may be used as a small reminder.

MODULE CODE

Any module requires a module code, three characters length. The module codes are delivered for free by the NO-SOD team. Internal modules (meaning specific modules for a company) must use the Inx module code. See <http://www.no-sod.org/certification.php> for details.

DATABASE

Implemented systems are using the following database components:

- System database: tod10.db
- Audit trail database: trail.db
- Module database: module code.db

Content of .db file provides (located in include/dbparameters):

- RDBMS type
- Host running the RDBMS
- Database name
- User to access the database
- Password (of the user)

Note

all .db files may point to the same host/database, for small configurations.

To use a database connection you must instantiate an object "nsdDB". Any access to this database must go through the instantiated object. Multiple database object may be instantiated.

File naming convention

Implemented systems are using the following database components:

File names must start with the three module code (always uppercase for "src" and "html" file types) characters.

Source code files have ".src" suffix (under /src/dev/)

Html template files have ".html" suffix (under /html/dev/)

Included files must have ".inc" suffix (under /include/ccl/...)

Library files must have ".lib" suffix (under /src/dev/)

All picture files must be stored in the directory "/pics/module code" where module code is lower case.

All icon files must be stored in the directory "/icons/module code" where module code is lower case.

All javascript files must be stored in the directory "/js" where the .js file name must contain the module code.

Variables naming convention

All registered session variables of a module must begin with the module code. Other variables must take care of reserved variables (lower/uppercase):

CERT, ACT, NSD*, SRC*, HTML*, DSP, MYSESS, MYSYSDB, MYTRAILDB, LOAD, CCL*, IP, FAULT, ERROR, POS, CUR_ACT, XMLDATA, XMLDATANAME, XMLINDVALUES, XMLINDNUMBER

SESSION VARIABLE - REGISTERED

To register a variable, put the variable in the \$_SESSION array. The variable must have the same name as the key in the array.

```
$_SESSION["XXXregisteredvar"]=$XXXregisteredvar;
```

To use a registered session variable, use the variable name directly (\$XXXregisteredvar)

To unregister, use the unset PHP function as follow: unset(\$_SESSION["XXXregisteredvar"])

RECEIPT DATA FROM HTML

Data are received through variable, transmitted with either GET or POST methods. To enable variable reception with the register_global parameter set to OFF (php.ini), use the following functions:

For POST data:

```
$nsdpostvars["srcFILENAME"]=array("var1","var2"...);
```

For GET data:

```
$nsdgetvars["srcFILENAME"]=array("var1","var2"...);
```

srcFILENAME is the source file called by the html get action. Var1, var2, etc are the list of the variable names to enable

OUTPUT GENERATION

To request from a source code file to merge an html template files with computed variables, and send the result to the client browser (as output):

Put the name of the html template file into the variable \$nsddspz

SOURCE CODE CALLING

From a source code file, you can link the process to the execution of another source code file by setting the variable \$sact to the name of the linked source code file.

From an html file, you can post, or get the \$act variable in order to define the source code file to call from the client browser. The only PHP file must stay the FLY.PHP

CERTIFICATE

Every transaction from the client browser to the server (source code file calling) must transmit the session certificate, stored in the \$cert variable.

e.g. [http://myserver/mynosodwebdirectory/fly.php?cert=\\$cert&act=XXXlist](http://myserver/mynosodwebdirectory/fly.php?cert=$cert&act=XXXlist)

Chapter 2. FLY.PHP - RUNTIME EXECUTION WORKFLOW

Every call from the browser to the HTTP server goes through the file called 'FLY.PHP'. This script manages the execution of the process within the NO-SOD Framework environment. By calling the FLY.PHP at every process of your web application, you take directly benefit of the framework security and integrated development environment.

Explanation step by step of the FLY process:

STEP 1: Set up environment variables

- nsdmailing

values = 0 (Inactive) or 1 (Active)

When activated, every call to the mailing functions provided in the framework will be executed. Set this to 0 if you do not have mail infrastructure, such a SMTP server.

- load

values = 0 or 1 or 2

This variable defines how the framework will find the source code and html files required during a module execution.

- 0 means use the web server file system
- 1 means use the system database
- 2 means use the web server development file system.

The major difference between load 0 and 2, is that with load 2, the documents loaded are not kept in memory for next calls.

This is easier to develop or debug in load 2, avoiding to logoff from the module at each trial. This is correct for all document types except LIB files. LIB files are always kept in memory.

- nsdsqtrace

values = 0 (Inactive) or 1 (Active)

When activated, the framework generates SQL logs of every database access under the /logs directory.

- nsdver

This sets the framework version. It's used to retrieve LIB, INC, SRC and HTML files of the current framework at runtime.

- nsddspz

This is the final output variable. Developers set this variable to the name of their html file to output, within the SRC code.

The FLY will use this variable in one of the latest steps to see if there is an output to process or not.

STEP 2: Including external files

Including external files

- nsddms.inc

This is main part of the NO-SOD Framework. It contains the management of the documents, database and session objects types.

These three towers cannot be loaded through the documents engines, because they are the mechanism of documents engine.

- ecpmo.d.inc

This is the Electronic Check Point module custom class file. Dedicated functions may be used by other module than ECP as well.

STEP 3: Get the client IP addresss

For audit trail, module security, and control reasons, the variable called 'ip' is set to the remote client ip address.

STEP 4: Check the current client browser window

If the client window browser name is not defined, set the name of this variable called 'nsdwin' to 'MAIN'. The 'MAIN' browser processes are going through every audit trails. Sometimes it's usefull that a subwindow browser of an application doesn't go through audit trail, particulary when this subwindow has for goal to check indefinitely if something has changed at the back-end side (like a big loop waiting the end of another process). E.g. btach job process are most of time requiring that kind of behaviour. To avoid having a trace of the subwindow hits every x seconds in your trail database, call the 'FLY' script with nsdwin as get or post variable, with a value not equal to 'MAIN'.

STEP 5: Start a new session if necessary

If the fly.php is called without the variable \$act defined, then the process start a new NO-SOD session.

To do so, here are some actions executed:

- Instantiate a new database object, and load in this object the system database parameters (coming from tod10.db file). Connect the database.
- Instantiate a new database object, and load in this object the audit trail database parameters (coming from trail.db file). Connect the database.
- Instantiate a new session object, and try to start it. If session started with the no-sod version, then:
 - Instantiate a new document object, and load the nosod.lib (NO-SOD library) file in the object.
 - Evaluate the library object content.
 - Register as session variables the system db object, the audit trail db object, the session and

the library objects.

- Set the \$act variable to "NSD1", the default action for new session (means login process).

In the case the session didn't start, the error returned will be "SIZ" => Sizing error, no more free session slots, try again later.

STEP 6: For non new sessions: take back session flow

Name the session with the certificate (\$cert), and start the php session.

Check if the session object is present. If not present, exit with error code "CER" => Certificate error, either the \$cert was wrong, or the session environment cannot be retrieved.

If the session is started successfully, declare all session variables as global (this exports all variables from the session array as global variables).

Verify if the remote client ip address has not suspiciously changed. If the IP is not same as in the session variable IP property, then exit with error code "TIP".

Connect the system database using the appropriate db object.

Evaluate every library object document defined in the library list property of the session object.

Declare all the variable from GET,POST and FILE php arrays, defined as expected in the source code execution (\$act) by the developer through \$nsdgetvars and \$ndpostvars arrays, as global variables.

Unserialize registered objects from the session object list property (cf keepobject method of the session object). This provide an effective way to set the Class definitions inside versionned documents, avoiding .INC flat versioning.

Track everything in the audit trail, and check if session is not timed out.

STEP 7: Source code extraction and execution

This process is looping indefinitely, unless no other action (source code extraction and execution) is requested. This check is done by verifying the \$act variable content.

If a source code set the \$act variable content to the name of another source code, then the process will be 'linked' to the requested action.

For each action requested, here comes the sequence:

- If not a NSD action (such login), verify if the user is logged in.
- Verify if the source code has already been loaded previously, if not load it in session variable. Except when \$load variable is set to 2. Then reload always the source code.
- When loading, verify execution rights: if the user do not have the X rights then generate an execution right error, and request the module .404 action source code.

- If the source code document object is not retrieved, exit with error code 'try to execute a non existing code'.
- Evaluates the source code object document content.

STEP 8: Output by merging html template.

The fly is checking the variable `$nsddspz`. If the variable is not empty, then retrieve the html object document name defined in the `$nsddspz` variable. The variable `$nsddspz` is set by the module developer in the action source code.

The html object document type are kept in memory as session variable as the `.src` (action source code) files. The loading process is exactly the same. The html document content is not directly evaluated, but is 'Evaluated and merged' with existing environment variables of the fly runtime execution. That's how these html files are templates.

Then the process send the merged result to the standard output.

STEP 9: Serializing objects

Every object having his class definition stored in a versionned source code file (so, not defined in `.INC` files) must be serialized and stored under a property of the session object, being an array of serialized objects. This is needed because if you register directly these objects as session variable, then at the next fly execution, the 'session start' statement will try to 're-life' these objects without having their class definition available (because stored inside source code document objects or library that have not yet been 're-life' themself).

STEP 10: Output error if any.

Check the variable `$error` and output the error definition. This comes when the process exited from one of the steps above. E.g. `SIZ,TIP,CER` errors.

Chapter 3. COMMONLY USED VARIABLES

Type: available variables

Location: nosod.lib / fly.php / nsddms.inc

Table 3.1. Variables list

Variable	Description
\$cert	Session certificate
\$mySess->uname	User name
\$act	Source code to execute (action)
\$nsddspz	Output html template (name of the display)
\$mySess	The session object
\$mysysDB	The system database object
\$mytrailDB	The audit trail database object
\$load	The loading mode for execution (dev: 2, database:1, file is load zero)
\$ip	Remote client browser's ip address
\$fault	Boolean value set at true when error occurred
\$error	The error message defined when \$fault is true
\$pos	Defined at zero where requested action requires to be logged in
\$cur_act	The current action (\$act) processed
\$xmldata	Global variable used for XML parsing
\$mySess->id	Session number
\$mySess->uid	The user id of the user logged in
\$mySess->fullname	The complete user name of the user logged in
\$mySess->modname	The current module initials
\$mySess->modversion	The current module version
\$mySess->modsrcver	The current source code version of the current module
\$mySess->modguiver	The current GUI version of the current module
\$mySess->nsdver	The current Framework version
\$mySess->ip	Remote client browser's ip address
\$mySess->start	Session start time stamp
\$mySess->liblist	Array of libraries currently used
\$mySess->objlist	Array of objects having class definition in libraries
\$mySess->cname	The remote browser's computer name
\$mySess->parameters	Array containing each parameter of the current module
\$mySess->userloggedin	Boolean value set to true when the user is logged in
\$mySess->profile	User profile of current user
\$mySess->message	Message to transmit to the user through the GUI

Chapter 4. HTMLGEN - Generating an html table dynamically

Type: standalone function

Location: nosod.lib

htmlgen(arg1, arg2)

Input:

- arg1 => array name:string
- arg2 => table template:string

Output: string (returned)

This function has for goal to generate automatically an html table content. The approach is to provide in 'arg1' the name of the array prepared by a previous process. This array must be two dimensions, where the first dimension is the row (starting at 0), and the second dimension is the column (starting at 0).

In arg2, you put as string a subpart of html table, identifying the row template. Inside the row template, put the characters '++vx++' at each place of a data, where x is the column number + 1 (means you start with ++v1++).

The function will return the complete html table generated. It's usefull when you generate a table from a database query, and that you do not know in advance how must row the table will contain. This function can be used also for generating html dropdowns and lists, as the template principle is the same.

Example: Suppose your html template file contain something like

```
"<TABLE>$INOK_INFO</TABLE>"
```

The source code generating the html output will be something like:

```
    "$query=$INODB->queryExec("SELECT NAME,FIRSTNAME FROM PEOPLE WHERE YEAROFBIRTH=20
while ($r = $INODB->queryFetch($query))
{
    $INOK_INFO[$i][0]=$r["NAME"];
    $INOK_INFO[$i][1]=$r["FIRSTNAME"];
    $i++;
}

eval(htmlgen("INOK_INFO", "<tr>
    <td><font face=\"Courier New\">v1</font></td>
    <td><font face=\"Courier New\">v2</font></td>
</tr>"));
"
```

Chapter 5. HTMLGENTABLE - Generating an html table dynamically with row effect

Type: standalone function

Location: nosod.lib

htmlgentable(arg1, arg2, arg3, arg4, arg5)

Input:

- arg1 => array name:string
- arg2 => html table template:string
- arg3 => background color variable name: string
- arg4 => first html color: string
- arg5 => second html color: string

Output: string (returned)

This function has for goal to generate automatically an html table content, as HTMLGEN, but is dedicated to produce table effects with one row on two having a different background color. The provide higher visibility in the table produced in the html output.

The approach is to provide in 'arg1' the name of the array prepared by a previous process. This array must be two dimensions, where the first dimension is the row (starting at 0), and the second dimension is the column (starting at 0).

In arg2, you put as string a subpart of html table, identifying the row template. Inside the row template, put the characters '+vx+' at each place of a data, where x is the column number + 1 (means you start with ++v1++).

In arg3, you must give the name of the background color variable defined in the row template as well. E.g. for a ++vbgcolor++ in the template, provide "bgcolor" in arg3.

In arg4 and arg 5, provide hexadecimal colors for rows background. E.g. #CCCCCC and #AAAAAA

The function will return the complete html table generated. It's usefull when you generate a table from a database query, and that you do not know in advance how must row the table will contain.

Example: Suppose your html template file contain something like

```
"<TABLE>$INOK_INFO</TABLE>"
```

The source code generating the html output will be something like:

```
    $query=$IN0DB->queryExec("SELECT NAME,FIRSTNAME FROM PEOPLE WHERE YEAROFBIRTH=2000")
    while ($r = $IN0DB->queryFetch($query))
    {
        $INOK_INFO[$i][0]=$r["NAME"];
        $INOK_INFO[$i][1]=$r["FIRSTNAME"];
    }
}
```

```
$i++;  
}  
  
eval(htmlgentable("INOK_INFO","<tr bgcolor=\"++vbgcolor++\">  
  <td><font face=\"Courier New\">v1</font></td>  
  <td><font face=\"Courier New\">v2</font></td>  
</tr>","#CCCCCC","#AAAAAA"));  
"
```

Chapter 6. The User Object definition

- class nsdUser

Location: nosod.lib

This class has for goal to provide user object definition to the application. Everything related to a user must be processed through this class, for reliability reasons.

Principle:

Every user definition is stored in the system database, as an XML document of type 'user'. So this class uses the nsdDoc class as storage engine, with document type 'usr'. Objects are stored in tables nsduser(master), nsdusers(versions) and nsduseri(exported indexes) tables.

Object properties

- uid - User identifier (unique number)
- uname - Username (part of the user e-signature, in the loggin)
- password - User's password (one way crypted value of the password)
- lastname - User's lastname
- firstname - USer's firstname
- initials - User's initials
- email - User's email address
- title - User's such IT Responsible, Administrator,...
- address - User's address
- zip - User's address zip code
- state - User's address state
- city - User's address city
- country - User's address country
- company - User's company name
- department - User's company department (eg Quality Assurance)
- businessunit - User's company businessunit (eg Admin services)
- phone - User's phone number
- gender - User's gender (Male/Female)
- active - User's account activated or disabled (boolean)
- start - Account starting date
- end - Account end date
- list - list of XML attirbutes exported as index

- defaultmodule - User's default module initials
- defaultmoduleversion - User's default module version
- error - error message when a method call has failed

Object methods

get([\$load=1])

- Retrieve a user definition from stored users objects, and put values in the object properties.
- Before calling this method, ensure to set the property 'uname' in the instantiated object.
- This method returns true if successfull, false on error.

add([\$load=1],[\$reason="ownership"])

- Store as new user object using object properties. This creates a new user. Set all object properties before calling the method.
- The method will manage also XML indexing automatically.
- Reason argument is used for audit trail.
- This method returns true if successfull, false on error.

delete([\$load=1],[\$reason="ownership"])

- Delete logically the user object. This delete a user.
- Before calling this method, ensure to set the property 'uname' in the instantiated object.
- Reason argument is used for audit trail.
- This method returns true if successfull, false on error.

undelete([\$load=1],[\$reason="ownership"])

- Undelete the user object. This undelete a deleted user.
- Before calling this method, ensure to set the property 'uname' in the instantiated object.
- Reason argument is used for audit trail.
- This method returns true if successfull, false on error.

-
-
-

•
•
•

update([\$load=1],[\$passupdate=false],[\$unlock=false],[\$reason="ownership"])

- Update an existing user object.
- Set the argument passupdate to true if you change the user password property.
- set the argument unlock to set back the active property of user to true.
- Reason argument is used for audit trail.
- This method returns true if successfull, false on error.

getuname(\$uid)

- Returns the username of the given user identifier (uid), or empty string if uid not found.

tsig(\$user,\$password,[\$userstorage=1])

- Tests the accuracy of an electronic signature. Userstorage argument is load mode.
- Returns true if correct, false if error.

login(\$user,\$password,[\$module="NSD"],[\$userstorage=1])

- Login a user in a requested module. Userstorage argument is load mode.
- This method returns true if successfull, false on error.

Chapter 7. The Group Object definition - class nsdGroup

Location: nosod.lib

This class has for goal to provide group object definition to the application. Everything related to user's group must be processed through this class, for reliability reasons.

Principle three types of group are defined:

- **SYS** : system groups. These are groups on which relies the developer module source code. The developer defines the groups of his module.
- **PUB** : public groups. Groups created by the system administrator. The administrator link or map the created public group to the existing system group.
- **PRI**: private groups. Groups created by users of the module. Some modules require the user can manage some private groups.

Linking users to groups.

Users can be attached to public or private groups, never directly to system groups.

Summary schema: user (n) -> (n) public or private group (1) -> (n) system group

Groups are stored inside the system database, in the table nsddgroup. Relationships between system groups and other groups are stored in the table nsddlink, as well as relationships between users and groups.

Object properties

- **gid** - Group identifier (unique number)
- **gname** - Group name (string)
- **description** - Group description (string)
- **module** - Module to which the group is associated (3 chars)
- **type** - Group's type (SYS/PUB/PRI) (3 chars)
- **error** - error message when a method call has failed

Object methods

add(\$gname,\$module,\$type,[\$gdesc=""],[\$reason="ownership"])

- Add a new group.
- Provide as first argument the groupname, then the module initials to which the group is associ-

ated, and finally the groupe type.

- Optional arguments are the group description, and the reason for adding the group.
- This method returns the unique group id if successfull, false on error.

getgid(\$gname,\$module,\$type)

- Retrieve the unique group id of given group name, group type and module.
- Arg1 is the group name.
- Arg2 is the module of the group.
- Arg3 is the group type (PRI/SYS/PUB)
- This method returns the unique group id if successfull, false on error.

getgname(\$gid)

- Retrieve the group name of a unique group id.
- This method returns the group name (string) if successfull, false on error.

update(\$oldname,\$gname,\$module,\$type,[\$gdesc=""])

- Rename a group of any type.
- Arg1 is the original name (existing).
- Arg2 is the new group name.
- Arg3 is the module initials.
- Arg4 is the group type.
- Arg5 (optional) is the new group description.
- This method returns the unique group id if successfull, false on error.

add-user(\$uname,\$gname,\$module,\$type,[\$reason="ownership"])

- Add a relationship between a user and a group of type PUB or PRI.
- Arg1 is the username of the account to add in the given group.
- Arg2 is the group name receiving the user.
- Arg3 is the module to which is associated the group receiving the user.

- Arg4 is the group type receiving the user.
- Arg5 (optional) is the electronic signature reason.
- This method returns the unique group id if successful, false on error.

de- luser(\$uname,\$gname,\$module,\$type,[\$reason="ownership"])

- Remove a relationship between a user and a group of type PUB or PRI.
- Arg1 is the username of the account to remove from the given group.
- Arg2 is the group name containing the user.
- Arg3 is the module to which is associated the group containing the user.
- Arg4 is the group type containing the user.
- Arg5 (optional) is the electronic signature reason.
- This method returns the unique group id if successful, false on error.

listusersofgroup(\$gname,\$module,\$type)

- Retrieve the users belonging to a PUB or PRI group.
- Arg1 is the group name.
- Arg2 is the module of the group.
- Arg3 is the group type (PRI/PUB)
- This method returns an single dimension array containing the list of usernames attached to the group if successful, false on error.

listgroupsofuser(\$uname,\$module,\$type)

- Retrieve all the private groups, or public groups (PRI or PUB) a user is attached to.
- Arg1 is the username.
- Arg2 is the module initials.
- Arg3 is the group type (PUB/PRI).
- This method returns an single dimension array containing the list of group names in which the username has been found if successful, false on error.

listgroupsofmodule(\$module,\$type)

- Retrieve the public or private groups associated to a module, and filter result with only the groups to which the current user belongs to.
- Arg 1 the the module initials.
- Arg2 is the group type you want the list return (PUB / PRI).
- This method returns a two dimensions array.
- The array contains the list of group unique id and group names to which the current logged in username belong to... if successfull, false on error.

userbelong2group(\$uname,\$gname,\$module,\$type)

- Returns true if a given username belongs to the given group (public or private), false if not.
- Arg1 is the username.
- Arg2 is the group name.
- Arg3 is the module related.
- Arg4 is the group type (PUB/PRI)

ad- dgroup(\$ngname,\$nmodule,\$ntype,\$gname,\$module,\$ type,[\$reason="ownership"])

- Creates a new relationship between a system group and a 'public or private' group.
- Arg1 is the public or private group name.
- Arg2 is the module of the pub/pri group to add.
- Arg3 is the the group type to add (PRI/PUB)
- Arg4 is the system group name.
- Arg5 is the module of the system group.
- Arg6 must be set to "SYS" (system).
- Arg7 is optional, it's the e-sig reason.
- The method return false on error, or the system group id after successfull operation.

del- group(\$ngname,\$nmodule,\$ntype,\$gname,\$module,\$t ype,\$reason="ownership")

- Removes a public or private group association from a system group.
- Arg1 is the public or private group name.

- Arg2 is the module of the pub/pri group to remove.
- Arg3 is the the group type to remove (PRI/PUB)
- Arg4 is the system group name.
- Arg5 is the module of the system group.
- Arg6 must be set to "SYS" (system).
- Arg7 is optional, it's the e-sig reason.
- The method return false on error, or the system group id after successfull operation.

listsubgroupsof- group(\$gname,\$module,\$type,\$nmodule,\$ntype)

- Returns the list of groups (either private or public) associated to a system group.
- Arg1 is the system group name.
- Arg2 is the module initials.
- Arg3 must be set to 'SYS'.
- Arg4 is the module initials.
- Arg5 is the type of group list requested (PRI/PUB)
- Return false on error, or a single dimension array with the group names.

listgroupsmanagedbyuser(\$uname,\$module)

- Returns the private groups owned by a given user name, for a module.
- Arg1 is the username.
- Arg2 is the module initials.
- Return false on error, or a single dimension array with the group names.

listmastergidofuser(\$uname)

- Returns system groups and public/private groups (all group) to which a user is attached to (going through each public/private group of the user).
- Return false on error, or a single dimension array with the group names (SYS/PRI/PUB).

Chapter 8. HOW LIBRARY (.LIB) FILES ARE WORKING ?

Location: as source code files

Library files such nosod.lib have for goal to contain sets of functions and classes usable by the module source code. Isolating these functions and classes in a lib file provide an efficient way to avoid code redundancy.

What's the difference with a .INC file ?

The main difference is that .LIB content is managed by the framework like a source code document. This means the document is stored, versionned and extracted as the complete module code. Dot INC files are not managed by the document management engine of the framework. To benefit of the security with versionned source code, use .LIB files instead of Dot INC. Library files do not need php tags (`<?php ?>`), .inc files need these tags.

How to manage session variables containing objects having their definition stored in a .LIB file ?

This question comes from the fact PHP regenerates objects at the session start statement. When the session starts (the php session, not the no-sod session... the session start statement is executed at every server hit) php needs to have available the class definitons of objects re-generated. If not, objects are lost. With .LIB files, the problem is that the class definition stored in a document object, are only available after the session start (as they are also session variables). In order to provide the possibility to put class definitions in .LIB files, every object of a .LIB that you want to register a session variable must be processed by the keepobject() function. So, to register an object, just call keepobject() and give the object name as argument. If you ask yourself how is it possible, the mechanism serializes all objects passed through the keepobject function before the end of the FLY execution. The FLY unserialize these serialized objects kept as session variable at the beginning of the FLY execution. Refer the FLY documentation for more details. Note that the keepobject function is a method of the session object (defined in the nsddms.inc file).

How to create a lib file for my module ?

Your module lib file name must begin with your module initials, and be a .lib extension. You must store the lib file in src/dev directory, with .src files. In order to declare your lib file to be loaded automatically by the framework, at your module login, you must put the name of your lib file (without the .lib) extension in the nsdmodule table of the system database, in the column 'libname'.