PostNuke API Command Reference

Marco Canini

Jim McDonald

John Robeson

Gregor J. Rothfuss

Jan Schrage

PostNuke API Command Reference

by Marco Canini, Jim McDonald, John Robeson, Gregor J. Rothfuss, and Jan Schrage

V1.8 Edition
Published 12nd May 2002

Table of Contents

Chapter 1. Overview

1.1. What is the PostNuke API?

The PostNuke API is a set of functions for the PostNuke content system that allows developers to access key information and manage their specific content in an easy fashion.

1.2. Advantages of an API

There are a number of advantages to having an API for the PostNuke system. Primarily, using an API allows developers to write PostNuke-compliant code that they can guarantee will still work whenever PostNuke is upgraded. This is especially important for a system such as PostNuke as the core functionality of the system is at current very much in flux and is altering rapidly to meet the demands being placed upon it. Having a stable interface into the system is one way of ensuring that the core developers can continue their work on updating and optimising the PostNuke system without continually breaking the code that module developers have written.

Other reasons for using an API include the ability for developers to start working with PostNuke more quickly without needing to understand the internals of the system, to have standardised ways of obtaining and manipulating information, and

1.3. Disadvantages of an API

The main disadvantage of having an API is when you want to do something that it does not have a suitable function for. This problem is addressed by encouraging developers to submit their own suggestions for new API functions, and even the functions themselves, for introduction into the core (details on the procedures for doing this are below).

An API also adds overhead to the entire system, but the trade-off in stability and ease of development more than compensates for this.

1.4. Who should use this API?

The PostNuke API is primarily aimed at developers who wish to write modules for the PostNuke system. In addition, some theme designers might use these functions to provide advanced features within their theme.

1.5. Status of the API

The PostNuke API is currently in alpha. This means that the API may well have extra functions and arguments added to it. However, all of the functions as outlined in this document will work as described, and continue to work as described for the foreseeable future. Any future versions of the API will note where functions have been superceded or deprecated, and developers will have at least 6 months between any major changes in the API being implemented and the older functions being removed from the core, allowing suitable time for migration.

1.6. This Document

This document gives an overview of the PostNuke API, and also includes a reference for each official function currently available to developers.

1.7. Related Documents

Other documents that might be of use in conjunction with this guide are the Module Development Guide, the Theme Development guide, the Categorised Data Guide, and the Output Functions Guide (all yet to be written).

1.8. Suggestions and Updates

The PostNuke API is a work-in-progress. There will be many functions that are missing from the API that developers would like. To this end, if a developer has a request for a particular function then they can submit it to the PostNuke features request list on SourceForge at the PostNuke Homepage (http://sourceforge.net/project/post-nuke). The same system can be used for sending in updates of current functions or new functions. If you are a developer and currently either globals directly or accessing one of the core tables to obtain or update information then please consider sending in a feature request so that the PostNuke team can develop an API to carry out the work instead. This will ensure that your code is more likely to continue working through future versions of PostNuke without modification, and that

Please note that the main requirements for the core PostNuke API are stability and a relatively small footprint. Due to this it is possible that your request for a new or updated function will get refused on the grounds that it is too specific or can easily be built from other core API functions. In such situations the PostNuke team will always try to provide a simple alternative, but please remember that submission of a new or updated part of the API does not guarantee inclusion.

Chapter 2. API Breakdown

The PostNuke API is broken down in to a number of different areas. These areas are as follows:

2.1. Users

This area covers everything related to users. Confirming that a user is logged in, getting a user-specific configuration parameter, or logging a user into the PostNuke system are examples of functions that fall into this area.

2.2. Modules

This area covers everything related to modules. Seeing if a particular module is available, setting a module-specific configuration parameter, or loading in a particular module API are examples of functions that fall into this area.

2.3. Security

This area covers everything related to security. Checking a user's ability to carry out a particular action and generating unique request identifier keys to avoid spoof and repeat attacks are examples of functions that fall into this area.

2.4. Sessions

This area covers everything related to HTTP sessions. Setting session variables, creating session cookies to pass to the user's web browser, and initialising the PHP settings to allow correct implementation of the allowable security measures are examples of functions that fall into this area.

2.5. Variables

This area covers everything related to handling of variables. Cleaning and sanitising user input, correctly escaping information to be stored in a database, and ensuring that filenames do not have operating-system-unfriendly characters in them are examples of functions that fall into this area.

2.6. Output

This area covers everything related to output. Creating a table of information, adding a drop-down list to a form, and printing a page for the user are examples of functions that fall into this area.

2.7. Database

This area covers everything related to database. Initialising a database connection, getting a handle to a database, and obtaining a list of tables that the database contains are examples of functions that fall into this area.

2.8. Miscellaneous

This area covers any number of other functions that are useful but do not belong to any particular group. Loading the current user theme, getting status messages from the previously run command, and carrying out HTTP redirects to other pages are examples of functions that fall into this area.

Chapter 3. Notes on Programming with the API

3.1. Use of *void*

Throughout the API reference, use is made of a type *void*, especially for specific return values. PHP does not in itself have a type of *void*, it is used in this document to refer to an unset value. It's very important for you to understand this difference since the the void parameter is heavily used by the exception handling system on which PostNuke API functions are based.

For example, `pnModAPIFunc()` takes parameters of a module name, type, and function, works out which actual module function to call; it then calls the module function and passes back the return value as its own return value. The problem with this is that if `pnModAPIFunc()` returned *false* for a failure to find the specified function then to the developer this would be indistinguishable from the function being found and itself returning false. *void* return values can be checked with the PHP `isset()` function, as shown below:

```
$articles = pnModAPIFunc('News', 'user', 'getarticles');
if (!isset($articles) && pnExceptionMajor() != PN_NO_EXCEPTION) {
  // pnModAPIFunc() failed
  return; // throw back exception
}
if ($articles == false)
  // getarticles failed
} else {
  // getarticles succeeded, data in $articles
}
```

Throughout the PostNuke API reference the return values of *void* and *false* are distinct. *false* is returned when an API call functioned correctly but returns a negative response. *void* is returned when an API call has internal problems and raises an exception. Note that this is a general rule, and there are a few exceptions out of necessity. These exceptions are noted in the reference documentation for the relevant function.

Chapter 4. API Reference

The following pages have a complete PostNuke API reference. Any functions that are not listed in this section are to be considered internal and should not be used by developers.

pnBlockGetInfo

Name

`pnBlockGetInfo` — get block information

Synopsis

```
array pnBlockGetInfo( int bid );
            BAD_PARAM
            DATABASE_ERROR
            ID_NOT_EXIST
```

Description

`pnBlockGetInfo()` returns an array of Block information.

Parameters

bid

The id of this block in the PostNuke system.

Return Values

This function returns an array of block information or *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises ID_NOT_EXIST if the block is unknown.

Notes

The block information array contains the following items:

bkey

title

: The display title of the block

content

: The container for block content. Serialized using `pnBlockVarsFromContent()` and `pnBlockVarsToContent()`.

url

: The url for blocks that depend on URLs, like a rss headlines block.

position

: The position of the block, currently one of 'l' (left), 'r' (right), 'm' (middle). This is likely to change in later versions to make placement more flexible.

weight

: The weight of the block, used for vertical placement of blocks.

active

: The status of a block, '1' for active, '0' for inactive.

refresh

: Indicates if a block should be refreshed. Examples would be headline blocks that fetch external content regularly.

last_update

: The date and time of the last refresh of the block.

language

: The language of the block.

mid

: The ID of the module a block belongs to. '0' indicates a block belonging to the core.

Examples

```
// Get information on block
$blockinfo = pnBlockGetInfo($bid);
```

pnBlockLoad

Name

`pnBlockLoad` — load a block

Synopsis

```
bool pnBlockLoad( string modname string block );
            BAD_PARAM
            DATABASE_ERROR
            ID_NOT_EXIST
            MODULE_FILE_NOT_EXIST
```

Description

`pnBlockLoad()` loads a block into the PostNuke system.

Parameters

modname

The well-known name of a module to load the block from. This parameter can be left empty, or set to 'Core' to load core blocks.

block

The name of the block to load

Return Values

This function returns *`true`* if the module loaded successfully, and *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises ID_NOT_EXIST if the block is unknown. This function raises MODULE_FILE_NOT_EXIST if the block file doesn't exist.

Notes

The PostNuke API keeps track of what blocks have been loaded, so multiple calls to `pnBlockLoad()` with the same parameters will return *`true`* each time.

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

This function does not display the block automatically, this must be carried out by a separate call to `pnBlockShow()`.

Examples

```
// Load the past article block from the News module
if (!pnBlockLoad('News', 'past')) {
  die('Could not load past articles block');
}
```

See Also

`pnBlockShow()`

pnBlockShow

Name

`pnBlockShow` — set configuration variable

Synopsis

```
string pnBlockShow( string modname, string block, array blockinfo );
            BAD_PARAM
            DATABASE_ERROR
            ID_NOT_EXIST
            MODULE_FILE_NOT_EXIST
```

Description

`pnBlockShow()` gets the output of a block by calling its 'block_display' function.

Parameters

modname

The well-known name of a module to display the block from. This parameter can be left empty, or set to 'Core' to display core blocks.

block

The name of the block to load

blockinfo

An associative array of parameters to pass to the block display function. The exact number and type of parameter is block-dependent.

Return Values

This function returns the output of the block display function is if it succeeds, or *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises ID_NOT_EXIST if the block is unknown. This function raises MODULE_FILE_NOT_EXIST if the block file doesn't exist.

Notes

TODO: nail down the block standards. Note that the PostNuke block standards state that blocks called in this way have to return output themselves. Especially, they need to implement the '{$modname}_{$block}block_display' function.

Examples

`TODO`

See Also

`pnBlockLoad()`

pnConfigGetVar

Name

`pnConfigGetVar` — get configuration variable

Synopsis

```
mixed pnConfigGetVar( string name );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnConfigGetVar()` obtains a configuration variable from the PostNuke system.

Parameters

name

The name of the configuration variable to obtain

Return Values

This function returns the requested variable if the variable exists. If the variable does not exist then this function will return *`void`*. This function also returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

The configuration variables available with this release of the API are as follows:

debug

: Level of debugging required. 0 is no debugging, 1 is normal debugging.

Version_Num

: Version of this PostNuke system

Version_ID

: ID of this PostNuke system; currently always 'PostNuke'

Version_Sub

: Sub-ID of this PostNuke system

startpage

: Name of initial module for this PostNuke system to load

adminmail

: Email address of site administrator

sitename

: Name of this PostNuke system

slogan

: Slogan of this PostNuke system

timezone_offset

: Timezone offset of this PostNuke system, in hours from GMT-12

seclevel

: Security level of this site; one of 'high', 'medium', or 'low'

banners

: Flag to state whether advertising banners are shown; '0' for no or '1' for yes

language

: Code of default language for this PostNuke system

locale

: Locale for this PostNuke system

CensorMode

: Flag to state whether censoring is active; '0' for no or '1' for yes

CensorList

Array of words that are censored by this PostNuke system

Other variables will be made available in future releases of the API.

Examples

```
// Get locale for this site
$locale = pnConfigGetVar('locale');
```

See Also

`pnConfigSetVar()`

pnConfigSetVar

Name

`pnConfigSetVar` — set configuration variable

Synopsis

```
bool pnConfigSetVar( string name, string value );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnConfigSetVar()` sets a configuration variable on the PostNuke system.

Parameters

name

The name of the configuration variable to set

value

The value to set the configuration variable to

Return Values

This function currently returns *`false`* due to not being implemented.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

This function is not currently implemented.

Examples

```
// Set locale for this site
if (!pnConfigSetVar('locale', 'en_GB')) {
    die('Error setting configuration variable');
}
```

See Also

`pnConfigGetVar()`

pnConfigDelVar

Name

`pnConfigDelVar` — delete a configuration variable

Synopsis

```
bool pnConfigDelVar( string name );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnConfigDelVar()` deletes a configuration variable from the PostNuke system.

Parameters

name

The name of the configuration variable to delete

Return Values

This function currently returns *`false`* due to not being implemented.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

This function is not currently implemented.

Examples

```
// Remove locale setting
if (!pnConfigDelVar('locale')) {
  die('Error deleting configuration variable');
}
```

See Also

`pnConfigGetVar()`, `pnConfigSetVar()`

pnDBGetConn

Name

`pnDBGetConn` — get database connection

Synopsis

```
array pnDBGetConn(void);
```

Description

`pnDBGetConn()` obtains database connection handles for direct querying of the PostNuke database.

Return Values

This function returns an array of database connection values. At current the only active value is the first one, which should be used for all database interaction.

Notes

Future versions of the PostNuke API might supply extra connection information through this function, to allow for features such as replicated or failover databases.

Examples

```
// Get database connection
list($dbconn) = pnDBGetConn();
```

See Also

`pnDBGetTables()`

pnDBGetTables

Name

`pnDBGetTables` — get database tables

Synopsis

```
array pnDBGetTables(void);
```

Description

`pnDBGetTables()` obtains database table information for direct querying of the PostNuke database.

Return Values

This function returns an array of database table information.

Notes

At current this information is only sparsely documented; more documentation on this array and direct database access for developers is expected soon.

Examples

```
// Get database tables
$pntable = pnDBGetTables();
```

See Also

`pnDBGetConn()`

pnDBInit

Name

`pnDBInit` — initialise PostNuke database connection

Synopsis

```
bool pnDBInit(void);
```

Description

`pnDBInit()` initialises connections to the PostNuke database.

Return Values

This function returns *`true`* if the connections were successfully made, or *`false`* if the connections were not successfully made.

Notes

`pnDBInit()` is normally called from within `pnInit()`, and as such should not be called by developers.

See Also

`pnInit()`

pnExceptionFree

Name

`pnExceptionFree` — reset exception status

Synopsis

```
void pnExceptionFree(void);
```

Description

`pnExceptionFree()` makes a reset of current exception status.

Return Values

This function does not return a value.

Notes

You must always call `pnExceptionFree()` when you handle a caught exception or equivalently you don't throw the exception back to the callers chain.

See Also

`pnExceptionId()` `pnExceptionMajor()` `pnExceptionSet()` `pnExceptionValue()`

pnExceptionId

Name

`pnExceptionId` — return current exception identifier

Synopsis

```
string pnExceptionId(void);
```

Description

`pnExceptionId()` returns the string identifying the exception. The character string contains the ID for the exception (its PHP class name).

Return Values

This function returns the current exception identifier. If invoked when no exception was raised a *`void`* value is returned.

See Also

`pnExceptionFree()` `pnExceptionMajor()` `pnExceptionSet()` `pnExceptionValue()`

pnExceptionMajor

Name

`pnExceptionMajor` — return current exception major number

Synopsis

```
int pnExceptionMajor(void);
```

Description

`pnExceptionMajor()` allows the caller to establish whether an exception was raised, and to get the type of raised exception. The major PN_NO_EXCEPTION identifies the state in which no exception was raised.

Return Values

This function returns one of these values: PN_NO_EXCEPTION, PN_USER_EXCEPTION or PN_SYSTEM_EXCEPTION.

See Also

`pnExceptionFree()` `pnExceptionId()` `pnExceptionSet()` `pnExceptionValue()`

pnExceptionSet

Name

`pnExceptionSet` — raise an exception

Synopsis

```
void pnExceptionSet(void);
```

Description

`pnExceptionSet()` allows a function to raise an exception. The caller must supply a value for the major parameter. The major parameter can have one of the values PN_NO_EXCEPTION, PN_USER_EXCEPTION, or PN_SYSTEM_EXCEPTION. The value of the major parameter constrains the other parameters in the call as follows:

* If the major parameter has the value PN_NO_EXCEPTION, this is a normal outcome to the operation. In this case, both exception_id and param must be NULL. Note that it is not necessary to invoke `pnExceptionSet()` to indicate a normal outcome; it is the default behavior if the method simply returns.

* For any other value of major it specifies either a user-defined or system exception. The exception_id parameter is the identifier representing the exception type. If the exception is declared to have members, the param parameter must be the exception struct (PHP class) containing the members. If the exception has no members, param must be NULL.

Return Values

This function does not return a value.

See Also

`pnExceptionFree()` `pnExceptionId()` `pnExceptionMajor()` `pnExceptionValue()`

pnExceptionValue

Name

`pnExceptionValue` — return current exception value

Synopsis

```
void pnExceptionValue(void);
```

Description

`pnExceptionValue()` returns an object corresponding to this exception.

Return Values

This function returns the object that was passed as parameter to `pnExceptionSet()`. If invoked when no exception or an exception for which there is no associated information was raised, a *`void`* value is returned.

See Also

`pnExceptionFree()` `pnExceptionId()` `pnExceptionMajor()` `pnExceptionSet()`

pnGetBaseURI

Name

`pnGetBaseURI` — get base URI for PostNuke

Synopsis

```
string pnGetBaseURI( );
```

Description

`pnGetBaseURI()` returns the base URI of PostNuke.

Return Values

This function returns the base URI of PostNuke.

Notes

URI is the path part of an URL, for instance in the URL 'http://www.postnuke.com/stuff/index.php' the URI would be '/stuff/index.php'.

Examples

```
// get base URI
$path = pnGetBaseURI();
```

See Also

`pnGetBaseURL()`

pnGetBaseURL

Name

`pnGetBaseURL` — obtain base URL for this site

Synopsis

```
string pnGetBaseURL(void);
```

Description

`pnGetBaseURL()` obtains the base URL for the site. The base url is defined as the full URL for the site minus any file information *i.e.* everything before the 'index.php' from your start page.

Return Values

This function returns the base URL of the site.

Notes

The last character in the string returned by `pnGetBaseURL()` is always '/'

`pnGetBaseURL()` replaces the old *`nukeurl`* global variable.

pnGetStatusMsg

Name

`pnGetStatusMsg` — obtain status message

Synopsis

```
string pnGetStatusMsg(void);
```

Description

`pnGetStatusMsg()` obtains the last status message posted for this session. The status message exists in one of two session variables: 'statusmsg' for a status message, or 'errormsg' for an error message. If both a status and an error message exists then the error message is returned.

Return Values

This function returns the value of the last status message posted, or *`void`* if no status message exists.

Notes

`pnGetStatusMsg()` is a destructive function - it deletes the two session variables 'statusmsg' and 'erorrmsg' during its operation.

See Also

`pnSessionSetVar()`

pnInit

Name

`pnInit` — initialise PostNuke

Synopsis

```
void pnInit(void);
```

Description

`pnInit()` initialises the PostNuke system. It loads the required includes, sets up sessions and database connections, and obtains the system configuration.

Return Values

`pnInit()` does not return a value; if the function fails somewhere it will exit.

Notes

In future `pnInit()` might give a return value, to make it easier to optionally include PostNuke functionality in third-party scripts.

pnModAPIFunc

Name

`pnModAPIFunc` — execute a module API function

Synopsis

```
mixed pnModFunc( string modname, string type, string func, array args );
            BAD_PARAM
            MODULE_FUNCTION_NOT_EXIST
```

Description

`pnModFunc()` calls a specific module API function.

Parameters

modname

The well-known name of a module to execute a function from

type

The type of function to execute; currently one of 'user' or 'admin'

func

The name of the module API function to execute

args

An associative array of arguments to pass to the module API function. The exact number and type of arguments is function-dependent.

Return Values

This function returns whatever the return value of the resultant function is if it succeeds. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises MODULE_FUNCTION_NOT_EXIST if the module function doesn't exist. This function throw back exceptions raised by module function.

Notes

Before calling this function you MUST load the module API with `pnModAPILoad`.

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Call News module user function getarticles with argument 'id' set to 3
$articles = pnModAPIFunc('News',
                         'user',
                         'getarticles',
```

```
                        array('id' => 3));
if (!isset($articles)) {
  // pnModAPIFunc() failed
} elseif ($articles == false)
  // getarticles failed
} else {
  // getarticles succeeded, data in $articles
}

// Call Permissions admin function list with no arguments
$list = pnModAPIFunc('Permissions',
                     'admin',
                     'list');
if (!isset($list)) {
  // pnModAPIFunc() failed
} elseif ($list == false)
  // list failed
} else {
  // list succeeded, data in $list
}
```

See Also

`pnModGetVar()`, `pnModAPILoad()`, `pnModFunc()`

pnModAPILoad

Name

`pnModAPILoad` — load a module API

Synopsis

```
int pnModAPILoad( string modname, string type );
            BAD_PARAM
            DATABASE_ERROR
            MODULE_FILE_NOT_EXIST
            MODULE_NOT_EXIST
```

Description

`pnModAPILoad()` loads extra functions into the PostNuke system to extend its abilities.

Parameters

modname

The well-known name of a module to load

type

The type of module functions to load; currently one of 'user' or 'admin'.

Return Values

This function returns *`true`* if the module loaded successfully. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises MODULE_FILE_NOT_EXIST if a module file doesn't exist. This function raises MODULE_NOT_EXIST if the module doesn't exist.

Notes

The PostNuke API keeps track of what modules have been loaded, so multiple calls to `pnModAPILoad()` with the same parameters will return *`true`* each time.

This function does not load in the relevant display functionality for this module automatically, this must be carried out by a separate call to `pnModLoad()`.

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// See if News module is available
if (pnModAvailable('News')) {
  // Load the user API for the News module
  if (!pnModAPILoad('News', 'user')) {
    die('Could not load News module API');
  }
}
```

See Also

`pnModLoad()`, `pnModGetVar()`

pnModAvailable

Name

`pnModAvailable` — check if a module is available

Synopsis

```
bool pnModAvailable( string modname );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnModAvailable()` confirms whether a module is available to be used within the PostNuke system.

Parameters

modname

The well-known name of a module to check

Return Values

This function returns *`true`* if the module is available for use, *`false`* if the module is not available for use. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

A module that is loaded into the PostNuke system but which is inactive or uninitialised is defined as not available for use.

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// See if News module is available
if (pnModAvailable('News')) {
  // Load the News module
  if (!pnModAPILoad('News', 'user')) {
    die('Could not load News module API');
  }
}
```

pnModCallHooks

Name

`pnModCallHooks` — carry out hook operations for module

Synopsis

```
bool pnModCallHooks( string hookobject string hookaction string hookid string
extrainfo );
            BAD_PARAM
            DATABASE_ERROR
            MODULE_FILE_NOT_EXIST
            MODULE_FUNCTION_NOT_EXIST
            MODULE_NOT_EXIST
```

Description

`pnModCallHooks()` carries out hook operations for a module

Parameters

hookobject

the object the hook is called for - either ’item’ or ’category’

hookaction

the action the hook is called for - one of ’create’, ’delete’, ’transform’, or ’display’

hookid

the id of the object the hook is called for (module-specific)

extrainfo

extra information for the hook, dependent on hookaction

Return Values

This returns the hook output as *`string`*. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data. This function throw back exceptions raised by other functions.

Notes

Each hook for the module is called in turn. If the action is ’display’ the result is the concatenation of each hook output where the hook area is ’GUI’. Otherwise the result is the *`extrainfo`* returned by the last hook that is called.

The list of hook actions is constantly being refined. Check the source code for a current list. TODO: Have an official online hook action registry.

Examples

```
// Let any hooks know that we are displaying an item.  As this is a display
// hook we’re passing a URL as the extra info, which is the URL that any
// hooks will show after they have finished their own work.  It is normal
// for that URL to bring the user back to this function
$output->SetInputMode(_PNH_VERBATIMINPUT);
$output->Text(pnModCallHooks(’item’,
                             ’display’,
                             $tid,
                             pnModURL(’Template’,
                                      ’user’,
                                      ’display’,
                                      array(’tid’ => $tid))));
$output->SetInputMode(_PNH_PARSEINPUT);
```

See Also

`pnModRegisterHook()` `pnModUnregisterHook()`

pnModDBInfoLoad

Name

`pnModDBInfoLoad` — load database infor for a module

Synopsis

```
bool pnModDBInfoLoad( string module string directory );
            BAD_PARAM
            DATABASE_ERROR
            MODULE_NOT_EXIST
```

Description

`pnModDBInfoLoad()` loads the database table information for a module. This information is merged with the

Parameters

module

The well-known name of a module for which to load the database table information

directory

The directory name of the module for which to load the database table information, if known

Return Values

This function returns *`true`*. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises MODULE_NOT_EXIST if the module doesn't exist.

Notes

If there is no database table information present then it is assumed that the module uses tables defined elsewhere, or no database tables at all, hence the function always returns true.

Examples

```
// Load the table information for the News module
pnModDBInfoLoad('News'
// Get the table information, now including the news tables
$pntable = pnDBGetTables();
```

pnModDelVar

Name

`pnModDelVar` — delete a module variable

Synopsis

```
bool pnModDelVar( string module string name );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnModDelVar()` deletes a module-specific variable from the PostNuke system.

Parameters

module

The well-known name of a module for which to delete the variable

name

The name of the module variable to delete

Return Values

This function returns *`true`* if the deletion is successful. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Delete news numitems variable
if (!pnModDelVar('News', 'numitems')) {
  die('Error deleting News variable numitems');
}
```

See Also

`pnModGetVar()`, `pnModSetVar()`

pnModFunc

Name

pnModFunc — execute a module function

Synopsis

```
mixed pnModFunc( string modname, string type, string func );
            BAD_PARAM
            MODULE_FUNCTION_NOT_EXIST
```

Description

pnModFunc() calls a specific module function.

Parameters

modname

The well-known name of a module to execute a function from

type

The type of function to execute; currently one of 'user' or 'admin'

func

The name of the module function to execute

Return Values

This function returns whatever the return value of the resultant function is if it succeeds. This function returns *void* if an exception was raised.

Note that the PostNuke module standards state that module functions called in this way have to return specific values themselves. For more details on this, please see the PostNuke Module Development Guide.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises MODULE_FUNCTION_NOT_EXIST if the module function doesn't exist. This function throw back exceptions raised by module function.

Notes

Before calling this function you MUST load the module with `pnModLoad`.

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Display news
$return = pnModFunc('News', 'user', 'display');
```

See Also

`pnModGetVar()`, `pnModLoad()`, `pnModAPIFunc()`

pnModGetAdminMods

Name

`pnModGetAdminMods` — get list of administration modules

Synopsis

```
array pnModGetAdminMods(void);
            DATABASE_ERROR
```

Description

`pnModGetAdminMods()` obtains information on the modules in this PostNuke system that are active and have an administrative component to them.

Return Values

This function returns an array of module information arrays if the module is found. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

Each module array contains the same set of items as a call to `pnModGetInfo()`

Examples

```
// Get list of available modules with administrative interfaces
$modarray = pnModGetAdminMods();
if (!$modarray) {
  die('Error getting list of administrative modules');
}
```

See Also

`pnModGetInfo()`, `pnModGetUserMods()`

pnModGetIDFromName

Name

`pnModGetIDFromName` — get module id from name

Synopsis

```
int pnModGetIDFromName( string module );
            BAD_PARAM
            DATABASE_ERROR
            MODULE_NOT_EXIST
```

Description

`pnModGetIDFromName()` obtains a module ID in this PostNuke system.

Parameters

module

The well-known name of a module for which to obtain the module ID

Return Values

This function returns the module ID for the given module. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises MODULE_NOT_EXIST if the module doesn't exist.

Notes

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Obtain ID for News module
$modid = pnModGetIDFromName('News');
```

See Also

`pnModGetInfo()`

pnModGetInfo

Name

`pnModGetInfo` — get module info

Synopsis

```
array pnModGetInfo( int modid );
            BAD_PARAM
            DATABASE_ERROR
            ID_NOT_EXIST
```

Description

`pnModGetInfo()` obtains information on a module in this PostNuke system.

Parameters

modid

The id of this module in the PostNuke system.

Return Values

This function returns an array of module information if the module is found. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises ID_NOT_EXIST if the *`modid`* is unknown.

Notes

The module array contains the following items:

name

The well-known name of the module

directory

The Filesystem directory in which this modules files exist

displayname

The display-friendly name for this module

description

A short description of this module's functionality

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Get information on News module
$modid = pnModGetIDFromName('News');
$modinfo = pnModGetInfo($modid);
```

See Also

`pnModGetIDFromName()`, `pnModGetUserMods()`, `pnModGetAdminMods()`, `pnModGetVar()`

pnModGetName

Name

`pnModGetName` — get name of current top-level module

Synopsis

```
string pnModGetName( );
```

Description

`pnModGetName()` returns the name of the current top-level module.

Return Values

This function currently returns the name of the current top-level module, *`false`* if not in a module.

Notes

Examples

```
// get module name
$name = pnModGetName();
```

pnModGetUserMods

Name

`pnModGetUserMods` — get list of user modules

Synopsis

```
array pnModGetUserMods(void);
            DATABASE_ERROR
```

Description

`pnModGetUserMods()` obtains information on the modules in this PostNuke system that are active and have a user component to them.

Return Values

This function returns an array of module information arrays if the module is found. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

Each module array contains the same set of items as a call to `pnModGetInfo()`

Examples

```
// Get list of available modules with user interfaces
$modarray = pnModGetUserMods();
if (!$modarray) {
  die('Error getting list of user modules');
}
```

See Also

`pnModGetInfo()`, `pnModGetAdminMods()`

pnModGetVar

Name

`pnModGetVar` — get module variable

Synopsis

```
mixed pnModGetVar( string module, string name );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnModGetVar()` obtains a module-specific variable from the PostNuke system.

Parameters

module

The well-known name of a module from which to obtain the variable

name

The name of the module variable to obtain

Return Values

This function returns the requested variable if the variable exist. If the variable does not exist then this function will return *`void`*. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

A module's well-known name is the uniquely assigned name by which the module is know throughout the PostNuke community. The well-known name normally corresponds to the directory in which the module code is placed; module information can be found from any PostNuke system by going to the modules administration section. Examples of well-known names are 'News', 'FAQ', and 'Comments'.

This function does *not* get the configuration information of the module itself. To obtain information such as the module ID or its current status you must the other PostNuke API calls available.

Examples

```
// Get the version information for the 'News' module
$newsmodver = pnModGetVar('News', 'Version');
```

See Also

`pnModSetVar()`, `pnModDelVar()`

pnModLoad

Name

pnModLoad — load module

Synopsis

```
int pnModLoad( string modname, string type );
            BAD_PARAM
            DATABASE_ERROR
            MODULE_FILE_NOT_EXIST
            MODULE_NOT_EXIST
```

Description

pnModLoad() loads extra display functions into the PostNuke system to extend its abilities.

Parameters

modname

The well-known name of a module to load

type

The type of module functions to load; currently one of 'user' or 'admin'.

Return Values

This function returns *true* if the module loaded successfully. This function returns *void* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises MODULE_FILE_NOT_EXIST if a module file doesn't exist. This function raises MODULE_NOT_EXIST if the module doesn't exist.

Notes

The PostNuke API keeps track of what modules have been loaded, so multiple calls to `pnModLoad()` with the same parameters will return *`true`* each time.

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

This function does not load in the relevant API functionality for this module automatically, this must be carried out by a separate call to `pnModAPILoad()`.

Examples

```
// See if News module is available
if (pnModAvailable('News')) {
  // Load the News module
  if (!pnModLoad('News', 'user')) {
    die('Could not load News module');
  }
}
```

See Also

`pnModAPILoad()`, `pnModGetVar()`

pnModRegisterHook

Name

`pnModRegisterHook` — register a hook function

Synopsis

```
bool pnModRegisterHook( string hookobject string hookaction string hookarea string
hookmodule string hooktype string hookfunc );
            DATABASE_ERROR
```

Description

`pnModRegisterHook()` registers a hook function.

Parameters

hookobject

The hook object - either ’item’ or ’category’

hookaction

The action the hook is called for - one of ’create’, ’delete’, ’transform’, or ’display’

hookarea

The area of the hook (either ’GUI’ or ’API’)

hookmodule

The name of the hook module

hooktype

The name of the hook type (’user’ or ’admin’)

hookfunc

The name of the hook function

Return Values

This function returns *`true`* on success. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

The list of hook actions is constantly being refined. Check the source code for a current list. TODO: Have an official online hook action registry.

Examples

```
// Set up module hooks
if (!pnModRegisterHook(’item’,
                       ’display’,
                       ’GUI’,
                       ’Ratings’,
```

```
                        'user',
                        'display')) {
        return false;
    }
```

See Also

`pnModUnregisterHook()`

pnModSetVar

Name

`pnModSetVar` — set module variable

Synopsis

```
bool pnModSetVar( string module, string name, string value );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnModSetVar()` sets a module-specific variable on the PostNuke system. If the variable does not exist then it is created.

Parameters

module

The well-known name of a module for which to set the variable

name

The name of the module variable to set

value

The value to set the module variable with

Return Values

This function returns *`true`* if the update is successful. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Set the version information for the 'News' module
pnModSetVar('News', 'Version', 2);
```

See Also

`pnModGetVar()`, `pnModDelVar()`

pnModUnregisterHook

Name

`pnModUnregisterHook` — unregister a hook function

Synopsis

```
bool pnModUnregisterHook( string hookobject string hookaction string hookarea string
hookmodule string hooktype string hookfunc );
            DATABASE_ERROR
```

Description

`pnModUnregisterHook()` unregisters a hook function.

Parameters

hookobject

The hook object - either ’item’ or ’category’

hookaction

The action the hook is called for - one of ’create’, ’delete’, ’transform’, or ’display’

hookarea

The area of the hook (either ’GUI’ or ’API’)

hookmodule

The name of the hook module

hooktype

The name of the hook type (’user’ or ’admin’)

hookfunc

The name of the hook function

Return Values

This function returns *`true`* on success. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

The list of hook actions is constantly being refined. Check the source code for a current list. TODO: Have an official online hook action registry.

Examples

```
// Remove module hooks
if (!pnModUnregisterHook('item',
                         'display',
                         'GUI',
                         'Ratings',
                         'user',
                         'display')) {
    pnSessionSetVar('errormsg', _RATINGSCOULDNOTUNREGISTER);
}
```

See Also

`pnModRegisterHook()`

pnModURL

Name

`pnModURL` — create module URL

Synopsis

```
string pnModURL( string modname, string type, string func, array args );
            BAD_PARAM
```

Description

`pnModURL()` creates a PostNuke-compatible URL for a specific module function.

Parameters

modname

The well-known name of a module for which to create the URL

type

The type of function for which to create the URL; currently one of ’user’ or ’admin’

func

The actual module function for which to create the URL

args

An associative array of arguments. The exact number and type of arguments is function-dependent

Return Values

This function returns a module URL string if successful. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter.

Notes

For more information on well-known names of modules please refer to the documentation for `pnModGetVar()`

Examples

```
// Create a URL to the News ’view’ function with parameters ’sid’ set to 3
// and ’index’ set to ’0’
$url = pnModURL(’News’, ’user’, ’view’, array(’sid’ => 3, ’index’ => 0));
```

See Also

`pnModGetVar()`, `pnModFunc()`

pnRedirect

Name

`pnRedirect` — redirect to another page

Synopsis

```
void pnRedirect( string url, );
```

Description

`pnRedirect()` creates an absolute URL to a PostNuke webpage and sets the relevant HTTP header accordingly.

Parameters

url

A url to redirect to. The URL can have GET parameters appended to it in the normal fashion.

Notes

`pnRedirect()` can handle absolute as well as relative URLs if required. This is most often used in conjunction with other URL-generating functions such as `pnModURL`.

`pnRedirect()` carries out a HTTP1.1-compliant redirection. This function should not be used when output is also required, as the output will not be displayed.

Examples

```
// Redirect back to the homepage
pnRedirect('index.php');

// Redirect to a news page
pnRedirect(pnModURL('News',
                    'user',
                    'view',
                    array('sid' => 3, 'index' => 0)));

// Redirect off-site
pnRedirect('http://www.remotesite.com');
```

See Also

`pnModURL()`

pnSecAddSchema

Name

pnSecAddSchema — add security schema

Synopsis

```
void pnSecAddSchema( string component, string schema );
```

Description

pnSecAddSchema() adds a component schema into the PostNuke system. The component schema list is not used anywhere inside PostNuke itself, but can be displayed as part of a GUI function to aid in configuration of the PostNuke permissions system.

Parameters

component

The component to which this authorisation schema applies

schema

The authorisation schema

Return Values

This function returns *true* if the authorisation schema is added successfully, *false* if the schema is not added successfully (normally due to a duplicate component name), and *void* if there was an internal API error whilst attempting to add the schema.

Notes

A full description of the PostNuke authorisation system is beyond the scope of this document. For more in formation on the authorisation system and its use of realms, components, and instances, please refer to the PostNuke Authorisation System documentation.

Note that this function was previously known as addinstanceschemainfo(). That name is deprecated but still functional in this version of the PostNuke API; it is recommended that any calls that are currently made to addinstanceschemainfo() are changed to pnSecAddSchema().

Examples

```
// Add security schema for foo module
pnSecAddSchema('Foo::', 'Name of foo:Type of foo:foo ID');
```

pnSecAuthAction

Name

`pnSecAuthAction` — authorise attempted action

Synopsis

```
bool pnSecAuthAction( int realm, string component, string instance, int level int
uid(opt) );
            DATABASE_ERROR
```

Description

`pnSecAuthAction()` checks against the authorisation information held within the PostNuke system to decide if a particular level of authorisation can be allowed.

Parameters

realm

The realm that this authorisation is taking place in; currently always 0

component

The component for which this authorisation is taking place

instance

The instance for which this authorisation is taking place

level

The level of access which this authorisation requires to be allowed

uid

The uid of the user to check for authorisation

Return Values

This function returns *`true`* if the level of authorisation granted to the user for the given arguments is greater than or equal to the required level, *`false`* if the level of authorisation granted to the user for the given arguments is less than the required level. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data.

Notes

A full description of the PostNuke authorisation system is beyond the scope of this document. For more in formation on the authorisation system and its use of realms, components, and instances, please refer to the PostNuke Authorisation System documentation.

This function was previously known as `authorised()`. That name is deprecated but still functional in this version of the PostNuke API; it is recommended that any calls that are currently made to `authorised()` are changed to `pnSecAuthAction()`.

Examples

```
// See if use is authorised to access 'foo' module admin functions
if (pnSecAuthAction(0,
                    'Foo::',
                    'My Foo:Foos of the world:4',
                    ACCESS_ADMIN)) {
  // Yes
} else {
  // No
}
```

pnSessionDelVar

Name

`pnSessionDelVar` — delete session variable

Synopsis

```
bool pnSessionDelVar( string name );
```

Description

`pnSessionDelVar()` deletes a session-specific variable from the PostNuke system.

Parameters

name

The name of the session variable to delete

Return Values

This function returns *`true`* if the variable was deleted successfully, and *`void`* if there was an internal API error whilst attempting to delete the variable.

Notes

See Also

`pnSessionGetVar()`, `pnSessionSetVar()`

pnSessionGetVar

Name

`pnSessionGetVar` — get session variable

Synopsis

```
mixed pnSessionGetVar( string name );
```

Description

`pnSessionGetVar()` obtains a session-specific variable from the PostNuke system.

Parameters

name

The name of the session variable to obtain

Return Values

This function returns the requested variable if it exists, *`false`* if the variable does not exist, and *`void`* if there was an internal API error whilst attempting to obtain the variable.

Notes

Session variables are not guaranteed to exist from one process run to the next. Internal housekeeping might remove sessions that have been inactive for long periods of time, so do not use session variables to store long-term information.

See Also

`pnSessionSetVar()`, `pnSessionDelVar()`

pnSessionInit

Name

`pnSessionInit` — initialse session

Synopsis

```
bool pnSessionInit(void);
```

Description

`pnSessionInit()` initialises a new or current session, setting user cookies as needed and obtaining current session variables.

Return Values

This function returns *`true`* if the session initialisation finished successfully, *`false`* if the session initialisation did not finish successfully, and *`void`* if there was an internal API error whilst attempting to initialise the session.

Notes

This function is normally called from `pnInit()`, and should not be called by modules or any other developer processes.

See Also

`pnInit()`, `pnSessionSetup()`

pnSessionSetup

Name

`pnSessionSetup` — setup session

Synopsis

```
bool pnSessionSetup(void);
```

Description

`pnSessionSetup()` sets a number of PHP configuration variables to allow the PostNuke system to handle persistent user sessions across multiple HTTP connections.

Return Values

This function returns *`true`* if the session setup finished successfully, *`false`* if the session setup did not finish successfully, and *`void`* if there was an internal API error whilst attempting to set up the session.

Notes

This function is normally called from `pnInit()`, and should not be called by modules or any other developer processes.

See Also

`pnInit()`, `pnSessionInit()`

pnSessionSetVar

Name

`pnSessionSetVar` — set session variable

Synopsis

```
bool pnSessionSetVar( string name string value );
```

Description

`pnSessionSetVar()` sets a session-specific variable on the PostNuke system. If the variable does not exist then it is created.

Parameters

name

The name of the session variable to set

value

The value to set the session variable with

Return Values

This function returns `true` if the variable was set, and `void` if there was an internal API error whilst attempting to set the variable.

Notes

Session variables are not guaranteed to exist from one process run to the next. Internal housekeeping might remove sessions that have been inactive for long periods of time, so do not use session variables to store long-term information.

See Also

`pnSessionGetVar()`, `pnSessionDelVar()`

pnThemeLoad

Name

`pnThemeLoad` — load display theme

Synopsis

```
bool pnThemeLoad(void);
```

Description

`pnThemeLoad()` loads the display theme as specified by the user, and sets a number of global variables for backwards-compatibility with older display themes.

Return Values

This function loads `true` if the theme was loaded successfully, and `false` if the theme was not loaded successfully.

Examples

```
// Load in user theme
if (!pnThemeLoad()) {
  die('Problem loading theme');
}
```

pnUserGetAll

Name

`pnUserGetAll` — get basic information on all users

Synopsis

```
array pnUserGetAll(void);
```

Description

`pnUserGetAll()` obtains a set of basic information on all active users in the PostNuke system.

> **Warning**
>
> `pnUserGetAll()` is deprecated but still functional in this version of the PostNuke API. You shouldn't use it, since it'll be removed when PostNuke will reach the 1.0 version. Instead you are strongly encouraged to use the users module API `getall` function: $users = pnModAPIFunc('users', 'user', 'getall');

Return Values

This function returns an array of associative arrays. The array elements are referenced by UID for quick access to information on a specific user. Each array corresponds to a particular user, and has the following array members:

uname

The user name of the user

uid

The user ID of the user

name

The full name of this user

email

The email address of this user

url

The URL of this user

avatar

The avatar of this user

Notes

Only the *`uname`* and *`uid`* fields of each array are guaranteed to be populated; the other items may be empty.

Examples

```
// Get all users
$allusers = pnUserGetAll();

// Display username and uid for each user
foreach ($allusers as $user) {
        echo "User ID $user[uid] has user name $user[uname]\n";
}

// Display name for uid 5
echo "User ID 5 is called {$allusers[5]['name']}\n";
```

pnUserGetLang

Name

`pnUserGetLang` — get current language

Synopsis

```
string pnUserGetLang()(void);
            DATABASE_ERROR
```

Description

`pnUserGetLang()` obtains the name of the current language that this session is using for output display.

Return Values

This function returns the name of the language being used. This function returns *`void`* if an exception was raised.

Exceptions

This function doesn't raise exceptions but throws back exceptions raised from other functions.

Notes

The user does not actually have to be logged in for this function to operate correctly. If the user is not logged in then this function will return the system's default language instead.

Examples

```
// Get the user's current language
$lang = pnUserGetLang();
```

pnUserGetTheme

Name

`pnUserGetTheme` — get current theme

Synopsis

```
string pnUserGetTheme(void);
            DATABASE_ERROR
            UNKNOWN
```

Description

`pnUserGetTheme()` obtains the name of the current theme that this session is using for output display.

Return Values

This function returns the name of the theme that is being used. This function returns *`void`* if an exception was raised.

Exceptions

This function raises UNKNOWN if it can't find the theme. This function throw back exceptions raised by other functions.

Notes

This function takes into account per-category and per-story theme overrides.

The user does not actually have to be logged in for this function to operate correctly. If the user is not logged in then this function will return the system's default theme instead.

Examples

```
// Get the user's current theme
$theme = pnUserGetTheme();
```

pnUserGetVar

Name

`pnUserGetVar` — get user variable

Synopsis

```
mixed pnUserGetVar( string name string uid(opt) );
            BAD_PARAM
            DATABASE_ERROR
            ID_NOT_EXIST
            MODULE_FILE_NOT_EXIST
            MODULE_FUNCTION_NOT_EXIST
            MODULE_NOT_EXIST
            NO_PERMISSION
            NOT_LOGGED_IN
            UNKNOWN
            VARIABLE_NOT_REGISTERED
```

Description

`pnUserGetVar()` obtains a user-specific variable from the PostNuke system. By default it will obtain a variable for the current user; if the optional *`uid`* variable is supplied it will obtain the variable for that user instead.

Parameters

name

The name of the user variable to obtain

uid

The uid of the user to obtain this variable for

Return Values

This function returns the requested variable if either the user is logged in to the system or an existing user's ID is passed in to the function, and the variable exists. If the variable does not exist this function will return *`void`*. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises ID_NOT_EXIST if the user id is unknown. This function raises MODULE_FILE_NOT_EXIST if a authentication module file doesn't exist. This function raises MODULE_FUNCTION_NOT_EXIST if a authentication module function doesn't exist. This function raises MODULE_NOT_EXIST if the authentication module doesn't exist. This function raises NOT_LOGGED_IN if the user is not logged in. This function raises NO_PERMISSION if you don't have right permission to get variable

value. This function raises UNKNOWN for internal problems. This function raises VARIABLE_NOT_REGISTERED if the metadata of variable you are asking for is not registered.

Notes

The user-specific variables available with this release of the API are as follows:

email
: The user's email address

name
: The user's real name

theme
: The name of the user's theme

timezone_offset
: The user's timezone offset, in hours from GMT-12

uid
: The user's identification number, normally only used for internal purposes

uname
: The user's login name

url
: The user's URL

user_avatar
: The filename of the user's avatar

Other variables will be made available in future releases of the API. You can register new user variables with the register_user_var module API function included in Modules module. See the User Variable section in the PostNuke Module Development Guide for getting an explanation on user variables.

Examples

```
// Show information for logged in user
if (pnUserLoggedIn()) {
  $name = pnUserGetVar('name');
  if (isset($name)) {
    echo "Welcome to the site, $name";
  }
}
```

```
// Show information for specific user
$useremail = pnUserGetVar('email', 5);
$username = pnUserGetVar('name', 5);
if (!empty($useremail) && !empty($username)) {
    echo "$username has email address $useremail";
}
```

See Also

`pnUserSetVar()`

pnUserGetVars

Name

`pnUserGetVars` — get all user variables

Synopsis

```
array pnUserGetVars( string uid );
```

Description

`pnUserGetVars()` gets all user variables for a specified user identified by *`uid`*. `pnUserGetVars()` is deprecated because it can't be integrated with the new Dynamic User Data architecture. See the User Variable section in the PostNuke Module Development Guide for getting an overview of the new architecture.

Parameters

uid

The user identifier

Return Values

This function returns an associative array of user variables where keys represent variable names and values variable values.

Notes

Examples

```
// Get user id
$uid = pnSessionGetVar('uid');

// Get user variables
$vars = pnUserGetVars($uid);

$user_description = $vars['name'] . ', ' . $vars['email'];
```

See Also

`pnUserGetVar()` `pnUserSetVar()`

pnUserLoggedIn

Name

`pnUserLoggedIn` — check if user is logged in

Synopsis

```
bool pnUserLoggedIn(void);
```

Description

`pnUserLoggedIn()` ascertains whether or not the current user is logged in to the PostNuke system.

Return Values

This function returns *true* if the user is logged in to the PostNuke system, and *false* if the user is not logged in.

Examples

See example for pnUserGetVar() for example of use.

See Also

pnUserLogIn(), pnUserLogOut()

pnUserLogIn

Name

pnUserLogIn — log user in

Synopsis

```
bool pnUserLogIn( string uname, string pass, string rememberme );
            BAD_PARAM
            DATABASE_ERROR
            MODULE_FILE_NOT_EXIST
            MODULE_FUNCTION_NOT_EXIST
            MODULE_NOT_EXIST
```

Description

pnUserLogIn() attempts to log the user in to the PostNuke system with the supplied parameters.

Parameters

uname

The username of the user attempting to log in

pass

The password of the user attempting to log in

rememberme

If *true*, PostNuke will attempt to remember this user's credentials as long as possible within the bounds of the security policy that has been laid down by the site administrator.

Return Values

This function returns *true* if the user is successfully logged in, otherwise it returns *false*. This function returns *void* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function throw back exceptions raised by other functions.

Examples

```
// Attempt to log user in
if (!pnUserLogIn(pnVarCleanFromInput('username'),
                 pnVarCleanFromInput('password'),
                 pnVarCleanFromInput('rememberme')) {
  die('Bad username/password');
} else {
  // User logged on successfully
}
```

See Also

`pnUserLoggedIn()`, `pnUserLogOut()`

pnUserLogOut

Name

`pnUserLogOut` — log user out

Synopsis

```
bool pnUserLogOut(void);
            DATABASE_ERROR
```

Description

`pnUserLogOut()` attempts to log the user out of the PostNuke system.

Return Values

This function returns *`true`* if the logout attempt was successful or the user is already logged out. This function returns *`void`* if an exception was raised.

Exceptions

This function raises DATABASE_ERROR if an error occurs while querying data.

Examples

```
// Log the user out
pnUserLogOut();
```

See Also

`pnUserLoggedIn()`, `pnUserLogIn()`

pnUserSetVar

Name

`pnUserSetVar` — set user variable

Synopsis

```
bool pnUserSetVar()( string name, string value );
            BAD_PARAM
            DATABASE_ERROR
            ID_NOT_EXIST
            MODULE_FILE_NOT_EXIST
            MODULE_FUNCTION_NOT_EXIST
            MODULE_NOT_EXIST
            NOT_LOGGED_IN
            NO_PERMISSION
            UNKNOWN
            VARIABLE_NOT_REGISTERED
```

Description

`pnUserSetVar()` sets a user-specific variable on the PostNuke system.

Parameters

name

The name of a user variable to set

value

The value to set the named user variable to

Return Values

This function returns *`true`* if the update is successful, *`false`* if the variable validation fails. This function returns *`void`* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises ID_NOT_EXIST if the user id is unknown. This function raises MODULE_FILE_NOT_EXIST if a authentication module file doesn't exist. This function raises MODULE_FUNCTION_NOT_EXIST if a authentication module function doesn't exist. This function raises MODULE_NOT_EXIST if the authentication module doesn't exist. This function raises NOT_LOGGED_IN if the user is not logged in. This function raises NO_PERMISSION if you don't have right permission to set variable value. This function raises UNKNOWN for internal problems. This function raises VARIABLE_NOT_REGISTERED if the metadata of variable you are trying to update is not registered.

Notes

The user-specific variables available with this release of the API listed in the documentation for `pnUserGetVar()` See the User Variable section in the PostNuke Module Development Guide for getting an explanation on user variables.

Examples

```
// Update the user's email address
if (!pnUserSetVar('email', 'foo@bar.net')) {
  die('Problem setting user variable');
}
```

See Also

`pnUserGetVar()`

pnUserValidateVar

Name

`pnUserValidateVar` — validate user variable

Synopsis

```
bool pnUserValidateVar()( string name, string value );
            BAD_PARAM
            DATABASE_ERROR
            UNKNOWN
            VARIABLE_NOT_REGISTERED
```

Description

`pnUserValidateVar()` validates a user-specific variable on the PostNuke system.

Parameters

name

The name of a user variable to validate

value

The value to be validate against the named user variable

Return Values

This function returns *true* if the validation is successful, *false* if the validation fails. This function returns *void* if an exception was raised.

Exceptions

This function raises BAD_PARAM if you pass an invalid parameter. This function raises DATABASE_ERROR if an error occurs while querying data. This function raises UNKNOWN for internal problems. This function raises VARIABLE_NOT_REGISTERED if the metadata of variable you are asking for is not registered.

Notes

See the User Variable section in the PostNuke Module Development Guide for getting an explanation on user variables.

Examples

```
// Validate the user's email address
if (!pnUserValidateVar('email', $email)) {
  module_user_askdata(array('errmsg' => 'Invalid email'));
}
// Validate the user's timezone_offset address
if (!pnUserValidateVar('timezone_offset', $timezone_offset)) {
  module_user_askdata(array('errmsg' => 'Invalid timezone offset'));
}

pnUserSetVar('email', $email);
pnUserSetVar('timezone_offset', $timezone_offset);
```

See Also

`pnUserGetVar()`

pnVarCensor

Name

`pnVarCensor` — remove censored words from variable

Synopsis

```
mixed pnVarCensor( string var, string ... );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnVarCensor()` takes a variable number of *`var`* arguments and for each one examines it for words which are deemed offensive or otherwise not allowed to be displayed. These words are replaced with asterix marks to show that words have been removed.

`pnVarCensor()` tries to be intelligent in its attempt to remove censored words whilst not censoring words on the censor list that happen to be embedded in a larger word.

Return Values

If `pnVarCensor()` is only passed a single *`var`* argument then it returns the corresponding censored variable. If `pnVarCensor()` is passed multiple arguments then it returns an array of corresponding censored variables. This function returns *`void`* if an exception was raised.

Exceptions

This function doesn't raise exceptions but throws back exceptions raised from other functions.

Notes

`pnVarCensor()` uses the information provided in the configuration setting 'CensorList' as the basis of the words that it censors. It also looks for commonly derivations of the words used to try to avoid censoring. The system is also case-insensitive.

It is possible to censor information either before it is stored in the database or before it is displayed on-screen. The choice of which one to do is up to the developer, but the implications of the two different approaches should be

considered. By censoring information before putting it in to the database it is stored internally with censored marks, and it is possible that part of the meaning might be lost due to a bad choice of words in the censor list. By censoring information before it is displayed the ability to add items to the censor list and have them take effect retrospectively is possible, but there is an overhead involved each time that the information is displayed, and the uncensored text will be stored in the database.

Care should be taken to consider the effect of censorship, and if it should be applied to all information that is passed in by the user or if it should only be used in specific cases. It is also up to the developer to decide whether to accept the censored version of text or to return it to the user for editing.

Examples

```
// Obtain user comment and censor it
$comment = pnVarCleanFromInput('comment');
$comment = pnVarCensor($comment);

// See if user input contains censored words
$userinput = pnVarCleanFromInput('info');
if ($userinput != pnVarCensor($userinput)) {
        // Contains censored words
}
```

pnVarCleanFromInput

Name

`pnVarCleanFromInput` — obtain form variable

Synopsis

```
mixed pnVarCleanFromInput( string name, string ... );
```

Description

`pnVarCleanFromInput()` takes a variable number of *`name`* arguments and for each one obtains the variable from the input namespace. It removes any preparsing done by PHP to ensure that the string is exactly as expected, without any escaped characters.

`pnVarCleanFromInput()` also removes any HTML tags that could be considered dangerous to the PostNuke system's security.

Return Values

If `pnVarCleanFromInput()` is only passed a single *`name`* argument then it returns the corresponding variable. If `pnVarCleanFromInput()` is passed multiple arguments then it returns an array of corresponding variables.

Notes

Obtaining input variables from the global namespace, or from arrays such as HTTP_POST_VARS, is not supported and should never be done. `pnVarCleanFromInput()` is the only supported way of obtaining such variables.

Examples

```
// Obtain a single value
$id = pnVarCleanFromInput('id');

// Obtain a number of values
list($name, $number) = pnVarCleanFromInput('name', 'number');
```

pnVarPrepForDisplay

Name

`pnVarPrepForDisplay` — prepare variable for display

Synopsis

```
mixed pnVarPrepForDisplay( string var, string ... );
```

Description

`pnVarPrepForDisplay()` takes a variable number of *`var`* arguments and for each one carries out suitable escaping of characters such that when output as part of an HTML page the exact string is displayed.

Return Values

If `pnVarPrepForDisplay()` is only passed a single *`var`* argument then it returns the corresponding display-ready variable. If `pnVarPrepForDisplay()` is passed multiple arguments then it returns an array of corresponding display-ready variables.

Notes

Running `pnVarPrepForDisplay()` multiple times is cumulative, as is running a combination on `pnVarPrepForDisplay()` and `pnVarPrepHTMLDisplay()`, and is not reversible. It recommended that variables that have been returned from `pnVarPrepForDisplay()` are only used in output functions, and then discarded.

Examples

```
// Get a version of name and number ready to display
list($outname,$outnumber) = pnVarPrepForDisplay($name, $number);

// Print some output
$html = "Name is $outname<p>
         Number is $outnumber<p>
         Other info is " . pnVarPrepForDisplay($otherinfo) . "<p>";
```

See Also

`pnVarPrepHTMLDisplay()`

pnVarPrepForOS

Name

`pnVarPrepForOS` — prepare variable for operating system

Synopsis

```
mixed pnVarPrepForOS( string var, string ... );
```

Description

`pnVarPrepForOS()` takes a variable number of *`filename`* arguments and for each one carries out suitable escaping of characters such that when used as part of an operating system filename it is valid and as close to the original string as possible.

Return Values

If `pnVarPrepForOS()` is only passed a single *`var`* argument then it returns the corresponding operating-system-ready variable. If `pnVarPrepForOS()` is passed multiple arguments then it returns an array of corresponding operating-system-ready variables.

Notes

If the parameter passed to `pnVarPrepForOS()` contains characters that are illegal for that operating system then they will be replaced with a suitable character (normally '_'). The directory separator ('/' on Unix systems, '\' on Windows systems) is considered one of these, and will be replaced. As such only filenames, and not pathnames, should be passed to this function.

This function does not need to be run on the content of files, just their names.

Running `pnVarPrepForOS()` multiple times is cumulative, and is not reversible unless the developer knows how many times it has been run. It recommended that variables that have been returned from `pnVarPrepForOS()` are only used in operating system functions, and then discarded.

Examples

```
// Get a version of name and number ready to use as directory and file names
list($osname, $osnumber) = pnVarPrepForOS($name, $number);

// Open file $topdirectory/$name/$number
$fh = fopen(pnVarPrepForOS($topdirectory) . "$osname/$osnumber", 'w');
```

pnVarPrepForStore

Name

`pnVarPrepForStore` — prepare variable for database storage

Synopsis

```
mixed pnVarPrepForStore( string var, string ... );
```

Description

`pnVarPrepForStore()` takes a variable number of *`var`* arguments and for each one carries out suitable escaping of characters such that when inserted into a database the exact string is stored.

Return Values

If `pnVarPrepForStore()` is only passed a single *`var`* argument then it returns the corresponding database-ready variable. If `pnVarPrepForStore()` is passed multiple arguments then it returns an array of corresponding database-ready variables.

Notes

Running `pnVarPrepForStore()` multiple times is cumulative, and is not reversible unless the developer knows how many times it has been run. It recommended that variables that have been returned from `pnVarPrepForStore()` are only used in SQL functions, and then discarded.

Examples

```
// Get a version of name and number ready to store in the database
list($dbname, $dbnumber) = pnVarPrepForStore($name, $number);

// Create some database-ready SQL
$sql = "SELECT * from table
    WHERE col1 = '$dbname'
    AND col2 = '$dbnumber'
    AND col3 = '" . pnVarPrepForStore($otherinfo) . "'";
```

pnVarPrepHTMLDisplay

Name

`pnVarPrepHTMLDisplay` — prepare variable for display, preserving some HTML tags

Synopsis

```
mixed pnVarPrepHTMLDisplay( string var, string ... );
            BAD_PARAM
            DATABASE_ERROR
```

Description

`pnVarPrepHTMLDisplay()` takes a variable number of *`var`* arguments and for each one carries out suitable escaping of characters such that when output as part of an HTML page the exact string is displayed, except for a number of admin-defined HTML tags which are left as-is for display purposes.

Return Values

If `pnVarPrepHTMLDisplay()` is only passed a single *`var`* argument then it returns the corresponding display-ready variable. If `pnVarPrepHTMLDisplay()` is passed multiple arguments then it returns an array of corresponding display-ready variables. This function returns *`void`* if an exception was raised.

Exceptions

This function doesn't raise exceptions but throws back exceptions raised from other functions.

Notes

`pnVarPrepHTMLDisplay()` should be used with great care, as it does allow certain HTML tags to be displayed.

The HTML tags that will be displayed are those defined in the configuration variable *`AllowableHTML`*, which is set on a per-instance basis by the site administrator.

Running `pnVarPrepHTMLDisplay()` multiple times is cumulative, as is running a combination on `pnVarPrepHTMLDisplay()` and `pnVarPrepForDisplay()`, and is not reversible. It recommended that variables that have been returned from `pnVarPrepHTMLDisplay()` are only used in output functions, and then discarded.

Examples

```
 // Get the home and body text of a story (allow specified HTML tags through)
list($hometext,$bodytext) = pnVarPrepHTMLDisplay($hometext, $bodytext);

// Print some output - numer should not have
// HTML tags in it, so use pnVarPrepForDisplay rather than pnVarPrepHTMLDisplay
$html = "Home text is $hometext<p>
         Body text is $bodytext<p>
         Story number is " . pnVarPrepForDisplay($storynum) . "<p>
         Notes are " . pnVarPrepHTMLDisplay($bodytext) . "<p>";
```

See Also

`pnVarPrepForDisplay()`

pnVarValidate

Name

`pnVarValidate` — validate a variable

Synopsis

```
bool pnVarValidate( string var string type array args );
```

Description

`pnVarValidate()` returns *`true`* if the validation was successful, *`false`* if not

Parameters

var

The variable to validate

type

The type of validation to perform either email or url

args

An optional Array of arguments

Return Values

This function currently returns *`true`* on successful validation, and *`false`* if validation was unsuccessful.

Notes

args is unimplemented currently.

Examples

```
// validate email address
if (!pnVarValidate('john.doe@example.com', 'email')) {
    die('email address is incorrect format');
}

// validate url
if(!pnVarValidate('http://www.example.com','url')) {
    die('URL is invalid');
}
```

See Also