

Contents

1	NAME	2
2	DESCRIPTION	2
3	PUBLIC VARIABLES	2
4	PUBLIC FUNCTIONS	2

1 NAME

OpenSLX::Basics - implements basic functionality for OpenSLX.

2 DESCRIPTION

This module exports basic functions, which are expected to be used all across OpenSLX.

3 PUBLIC VARIABLES

%openslxConfig

This hash holds the active openslx configuration.

The initial content is based on environment variables or default values. Calling `openslxInit()` will read the configuration files and/or cmdline arguments and modify this hash accordingly.

The individual entries of this hash are documented in the manual of the *slxsettings*-script, so please look there if you'd like to know more.

%cmdlineConfig

This hash holds the config items that were specified via cmdline. This can be useful if you need to find out which settings have been specified via cmdline and which ones have come from a config file.

Currently, only the *slxsettings* script and some tests make use of this hash.

4 PUBLIC FUNCTIONS

openslxInit()

Initializes OpenSLX environment - every script should invoke this function before it invokes any other.

Basically, this function reads in the configuration and sets up logging and translation backends.

Returns 1 upon success and dies in case of a problem.

vlog(\$level, \$message)

Logs the given *\$message* if the current log level is equal or greater than the given *\$level*.

_tr(\$originalMsg, @msgParams)

Translates the english text given in *\$originalMsg* to the currently selected language, passing on any given additional *\$msgParams* to the translation process (as printf arguments).

N.B.: although it starts with an underscore, this is still a public function!

callInSubprocess(\$childFunc)

Forks the current process and invokes the code given in *\$childFunc* in the child process. The parent blocks until the child has executed that function.

If an error occurred during execution of *\$childFunc*, the parent process will cleanup the child and then pass back that error with an invocation of `die()`.

If the process of executing *\$childFunc* is being interrupted by a signal, the parent will cleanup and then exit with an appropriate exit code.

executeInSubprocess(@cmdlineArgs)

Forks the current process and executes the program given in *@cmdlineArgs* in the child process.

The parent process returns immediately after having spawned the new process, returning the process-ID of the child.

slxsystem(@cmdlineArgs)

Executes a new program specified by *@cmdlineArgs* and waits until it is done.

Returns the exit code of the execution (usually 0 if everything is ok).

If any signal (other than SIGPIPE) interrupts the execution, this function dies with an appropriate error message. SIGPIPE is being ignored in order to ignore any failed FTP connections and the like (we just return the error code instead).

cluck(), carp(), warn(), confess(), croak(), die()

Overrides of the respective functions in *Carp::* or *CORE::* that mark any warnings with 'ÃÃÃ' and any errors with '***' in order to make them more visible in the output.

checkParams(\$params, \$paramsSpec)

Utility function that can be used by any function that accepts param-hashes to check if the parameters given in *\$params* actually match the expectations specified in *\$paramsSpec*.

Each individual parameter has a specification that describes the expectation that the calling function has towards this param. The following specifications are supported:

* '!' - the parameter is required * '?' - the parameter is optional * 'm{regex}' - the parameter must match the given regex * '!class=...' - the parameter is required and must be an object of the given class * '?class=...' - if the parameter has been given, it must be an object of the given class

The function will confess for any unknown, missing, or non-matching param.

instantiateClass(\$class, \$flags)

Loads the required module and instantiates an object of the class given in *\$class*.

The following flags can be specified via *\$flags*-hashref:

acceptMissing [optional]

Usually, this function will die if the corresponding module could not be found (acceptMissing == 0). Pass in acceptMissing => 1 if you want this function to return undef instead.

pathToClass [optional]

Sometimes, the module specified in *\$class* lives relative to another path. If so, you can specify the base path of that module via this flag.

incPaths [optional]

Some modules live outside of the standard perl search paths. If you'd like to load such a module, you can specify one (or more) paths that will be added to @INC while trying to load the module.

version [optional]

If you require a specific version of the module, you can specify the version number via the *\$version* flag.

loadDistroModule(\$params)

Tries to determine the most appropriate distro module for the context specified via the given *\$params*.

During that process, this function will try to load several different modules, working its way from the most specific down to a generic fallback.

For example: when given *suse-10.3_x86_64* as distroName, this function would try the following modules:

Suse_10_3_x86_64

Suse_10_3

Suse_10

Suse

Base (or whatever has been given as fallback name)

The *\$params*-hashref supports the following entries:

distroName

Specifies the name of the distro as it was retrieved from the vendor-OS (e.g. 'suse-10.2' or 'ubuntu-8.04_amd64').

distroScope

Specifies the scope of the required distro class (e.g. 'OpenSLX::OSSetup::Distro' or 'vmware::OpenSLX::Distro').

fallbackName [optional]

Instead of the default 'Base', you can specify the name of a different fallback class that will be tried if no module matching the given distro name could be found.

pathToClass [optional]

If you require the distro modules to be loaded relative to a specific path, you can specify that base path via the *\$pathToClass* param.